NECパーソナルコンピュータ
**PC-9800シリーズ**

NEC

# Software Library

## N88-日本語BASIC(86)(MS-DOS版)5.0 リファレンスマニュアル

# Software Library

# N88-日本語BASIC(86)(MS-DOS版)5.0
## リファレンスマニュアル

AA811B

# はじめに

PC-9800シリーズパーソナルコンピュータには，標準プログラミング言語として$N_{88}$-日本語BASIC(86)が用意されています．

一方，MS-DOSの普及にともない，MS-DOS上で使用可能なBASIC言語のニーズも高まってきています．このようなニーズに答えるのが，この「$N_{88}$-日本語BASIC(86)(MS-DOS版)」です．

$N_{88}$-日本語BASIC(86)(MS-DOS版)は，$N_{88}$-日本語BASIC(86)と互換性のある，MS-DOS上で使用可能なプログラミング言語です．$N_{88}$-日本語BASIC(86)(MS-DOS版)で記述されたプログラムは，MS-DOS上で動作する$N_{88}$-日本語BASIC(86)(MS-DOS版)インタプリタ上で実行させることができます．

また，$N_{88}$-日本語BASIC(86)(MS-DOS版)で記述されたプログラムは，$N_{88}$-日本語BASIC(86)コンパイラでコンパイルすることによって，MS-DOS上で直接実行させることができます．$N_{88}$-日本語BASIC(86)コンパイラは，機能／処理能力とも$N_{88}$-日本語BASIC(86)(MS-DOS版)インタプリタとほぼ同等のものをもっており，コンパイラによってコンパイルされたプログラムは，インタプリタ上で実行される場合にくらべ，2～4倍の高速で動作します．

本書は，$N_{88}$-BASIC(86)(MS-DOS版)5.0の言語機能を，命令・関数単位で詳細に解説した文法説明書です．

$N_{88}$-BASIC(86)(MS-DOS版)インタプリタおよびコンパイラの操作法，動作環境，あるいは応用的な操作などについては，「$N_{88}$-日本語BASIC(86)(MS-DOS版)5.0ユーザーズマニュアル」を参照してください．

# このマニュアルの構成

本書は，次の3章および付録から構成されています．

### 第1章　BASICの文法

$N_{88}$-BASIC(86)(MS-DOS版)言語の文法・規則についての解説です．本BASICを利用する前には必ず本章を読み，その内容を理解・把握しておいてください．

### 第2章　命令リファレンス

$N_{88}$-BASIC(86)(MS-DOS版)で使用可能な命令・関数をアルファベット順に並べ，個々に詳細な解説をしています．本BASICの詳しい辞書として利用してください．

### 第3章　サンプルプログラム

相互に関連のある命令・関数を使用して作成したサンプルプログラム集です．実際にプログラムを作成する際の参考にしてください．

### 付　録

エラーメッセージとその対策，キャラクタコード表，予約語一覧などの付加情報・資料です．

なお，本書においては，キー操作や日本語入力の方法，各機能の応用的な説明などに関しては記述していません．これらの解説は，「$N_{88}$-日本語BASIC(86)(MS-DOS版)5.0ユーザーズマニュアル」に記載されていますので，そちらを参照してください．

# 目　次

## 第2章 命令リファレンス

## 第3章 サンプルプログラム



## 付 録



## 索 引

# 命令・関数の機能別索引

## コマンド



## グラフィック画面制御

## 一般命令



## テキスト画面制御

## 算術関数



## パック10進演算



## 1バイト系文字列操作

## 日本語文字列操作



## ファイル制御



## キー制御

## I/O ポート入出力制御



## チャイルドプロセス制御



## メモリ管理



## 機械語プログラム制御



## エラー制御

## 時刻／日付け制御



## プリンタ制御



## RS-232C コミュニケーションボード制御



## マウス制御

## GP-IB 制御



## サウンド制御



## ネットワーク制御(MS-NETWORKS)

## ネットワーク制御(BRANCH 4670)

命　令



関　数



## 電話制御

命　令



関　数

# $N_{88}$-日本語BASIC(86)(MS-DOS版)の特徴

$N_{88}$-日本語 BASIC(86)(MS-DOS 版)は，PC9800 シリーズに同梱され，現在広く使用されている $N_{88}$-日本語 BASIC(86)と言語上の互換性が高い言語です．

## $N_{88}$-日本語 BASIC(86)(MS-DOS 版)と $N_{88}$-日本語 BASIC(86)の違い．

1. $N_{88}$-日本語 BASIC(86)(MS-DOS 版)で記述されたプログラムは MS-DOS 上で動作する $N_{88}$-日本語 BASIC(86)(MS-DOS 版)インタプリタの制御下で動作します．
   また，$N_{88}$-日本語 BASIC(86)(MS-DOS 版)で記述されたプログラムは，$N_{88}$-日本語 BASIC(86)(MS-DOS 版)コンパイラでコンパイルすることにより MS-DOS 上で実行することができます．
2. ディスクファイルの形式が BASIC 形式のものから MS-DOS 形式に変わっています．
3. 日本語内部コードが JIS コード形式(KI/KO コード挿入式)からシフト JIS コード形式に変わっています．
4. プログラム実行時のメモリレイアウトが違います．
5. 次の機能は $N_{88}$-日本語 BASIC(86)(MS-DOS 版)では提供されません．
   ターミナルモード
   モニタモード
   カセットテープ入出力命令
   ライトペン制御命令
6. 次の機能があらたに提供されます．
   マウス制御命令
   チャイルドプロセス制御命令
   10 進演算機能

なお，$N_{88}$-日本語 BASIC(86)と $N_{88}$-日本語 BASIC(86)(MS-DOS 版)の機能差の詳細に関しては $N_{88}$-日本語 BASIC(86)(MS-DOS 版)5.0ユーザーズマニュアルを参照してください．

これらの相違点をのぞき言語の規則／プログラミングの考え方は $N_{88}$-日本語 BASIC(86)と同様です．

## $N_{88}$-日本語 BASIC(86)(MS-DOS 版)インタプリタの利点．

1. MS-DOS 上で動作する種々のアプリケーションプログラムと直接データ交換が行えます．
2. MS-DOS 上で動作する種々のアプリケーションプログラムと BASIC プログラムの連係動作が可能です．

3. MS-DOS が提供する種々のファンクション／ユーティリティを使用することができます.

これらの利点は $N_{88}$-日本語 BASIC(86)(MS-DOS 版)コンパイラでコンパイルされたプログラムについても同様です.

## $N_{88}$-日本語 BASIC(86)(MS-DOS 版)インタプリタとコンパイラの言語機能の違い.

1. コンパイラでコンパイルされたプログラムは MS-DOS 上で直接実行されますので，次に示すコマンド類は意味がなく，使用できません.
   AUTO, CONT, DELETE, EDIT, LIST, LLIST, LOAD, MERGE, NEW, RENUM, SAVE
2. プログラム動作時のメモリ環境の相違から，メモリレイアウトに関係する種々の命令に差があります.
3. コンパイラでコンパイルされたプログラムでは，データの型がコンパイル時に決定されます. したがって，動的に変数の型を変えることはできません.

これらの機能差に関する詳細については第1章および第2章を参照してください.

# 第1章

# BASICの文法

1

# 第1章　BASICの文法

## 1.　文

文は，BASIC が行う手続き（すなわちコンピュータに実行させるべき処理）を記述する最小単位です．

文は１つの命令からなります．そして命令は，命令の名前と，命令に与える情報（パラメータ）とからなります．パラメータには定数，変数，関数(「第１章　10.4 関数」参照)のほか，これらを組み合せた式などを指定することができます．

```
PRINT        1 2 3 4
└─────┘      └─────┘
命令の名前    パラメータ
└────────────────────┘
      命令／文
```

## 2.　行

行とは，プログラムモードで実行する場合において，メモリ中に格納(記憶)されるプログラムとして書かれる，行単位の文の集まりのことです．

行の先頭には行番号をつけます．１行（行番号も含めて）には255バイトの範囲で文の記述が可能です．

行は，基本的には１つの文からなりますが，複数の文を記述することも可能です．１行に複数の文を記述する場合，各文をコロン(：)で区切ります．これは複文(マルチステートメント)と呼ばれ，１行の範囲内で任意の数の複文が記述できます．

## 3.　行番号

行番号は１から65529までの整数で指定します．

行番号は行をメモリに格納したりディスクにセーブしたりする場合の格納順序を示すものです．

行番号をつけた行は，リターンキーの入力とともにメモリに格納されます．この方法で１行以上の行をメモリに格納したものがプログラムです．

プログラムの実行は行番号の小さいものから行われます．

行番号は分岐（プログラムの実行順序を変えること）の場合の分岐先として使用されるほか，LIST，DELETEなど〈行番号〉を対象とする命令のパラメータ（引数）として使われます．

**備考**：$N_{88}$-日本語BASIC(86)(MS-DOS版)インタプリタでは，行に行番号を付ける場合と付けない場合とでその動作が異なります．

行番号がなく，いきなり文が記述されている場合，リターンキーの入力によりその文は直ちに実行されます．

先頭に行番号が記述されている場合，その行はプログラムを構成する1つの行としてメモリに格納されます．

なお，コンパイラではあらかじめ行番号つきで作成しておいたディスク上のプログラムだけが処理の対象となります．

## 4.　BASICで使用できる文字と特殊記号

使用できる文字は次のとおりです．

- 文字
  - 1バイト系文字
    - 英大文字
    - 英小文字
    - 数字
    - カナ
    - 特殊記号
    - グラフィックキャラクタ
    - コントロールキャラクタ
  - 日本語文字（2バイト系日本語文字）
    - 2バイト系全角文字
    - 2バイト系半角文字
    - 2バイト系4分の1角文字

ただし,グラフィックキャラクタと日本語文字を同時に扱うことはできません(詳細は第2章のCONSOLEの説明を参照してください)．

1バイト系の英大文字と英小文字は，キーボード上から入力できる通常の英文字です．BASICでは，特定の場所(たとえば文字型定数の中…文字型定数に関しては「第1章　5.1　文字型定数」参照)を除き，英大文字と小文字とは同じ意味に解釈されます．

コントロールキャラクタは，スクリーンエディタに直接指示を与えるために使うもので，キーボード上の CTRL キーを押しながら英文字1文字のキーを押すことによって入力します．

日本語文字は，日本語処理システムによって入力する文字です．命令によっては使えないものもありますので注意してください．

これらの文字の詳細については，「付録B．コントロールコード表」，「付録C．キャラクタコード表」，あるいは各機種本体に添付されているハードウェアマニュアルの「漢字コード表」などを参照してください．

**■特殊記号の使い方**

算術演算子（+，−，＊，／）などの他にも特別な意味を持つ記号があります．ここでその意味をまとめて説明しておきます．なお，ここで説明しているのは1バイト系の特殊記号です．2バイト系日本語文字の中にも同様な記号がありますが，それらは文字型定数としてのみ使えるもので，特殊記号として使うことはできません．

⑴ **ピリオド（.）**

最後にBASICの処理の対象となった行（現在行，着目行などという）の行番号の値を示す記号で，命令中の〈行番号〉の代わりとして使用することができます．たとえば，新しい行を挿入した，エラーが発生したなどの行です．

例）　LIST　.

⑵ **マイナス（−）**

LIST，DELETEなどで行の範囲を指定するとき，何行から何行までという場合に使います．算術演算子の負号と同じ記号です．

例）　DELETE　100−200

⑶ **コロン（:）**

マルチステートメントの区切りとして使います．

例）　A=B+C : PRINT　A

⑷ **コンマ（,）**

パラメータが並ぶ場合その区切りとして使います．

例）　INPUT　A，B，C
　　　COLOR　7，，，0

⑸ **セミコロン（;）**

PRINTなどの出力文中でパラメータが並ぶ場合その区切りとして使います．

例）　PRINT　"A="；A

⑹ **アポストロフィ（'）**

REM（第2章中のREMの項参照）の代用として使います．

⑺ **クエッションマーク（?）**

PRINTの代用として使います．

⑻　**アスタリスク（＊）**

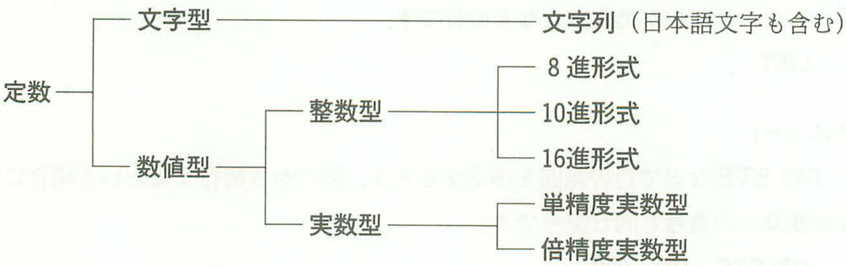
ラベル名の先頭につけます（「第1章　13. ラベル名」参照）.

⑼　**スペース**

原則として文字型定数に含まれているスペース以外のスペースは無視されますが，命令の名前の直後には必ずスペースを入れなければなりません.

## 5.　定数

定数とは，それぞれ固有の値を持ったデータのことで，プログラム中などに直接表記されるものです．定数は，次のように分類することができます．定数の表記法は，型によってそれぞれ異なります.




### 5.1　文字型定数

文字型定数とは，255バイト以内の文字をつなぎ，その前後をダブルクォーテーション（"）で囲んだ，文字列データのことです．文字列中には，1バイト系の英数字，カナ文字，特殊記号，グラフィックキャラクタならびに2バイト系日本語文字などを使用できます．なお，グラフィックキャラクタと2バイト系日本語文字を混在させることはできません.

BASICの命令のなかには，2バイト系日本語文字が使用できないものもあります(各命令参照)．また，(")を文字型定数中に直接記述することはできません．(")を文字型定数としたい場合，例のようにCHR $ (&H22) を使う方法を用います.

なお，文字型定数を算術演算に使うことはできません.

例）

```
"Good  Morning"
"1 2 3 4 5 6 7 8 9"
"パーソナル　コンピュータ"
"日本語 BASIC"
CHR $ (&H22)＋ "TEST"＋CHR $ (&H22)
```

### 5.2　数値型定数

数値型定数とは，算術演算を行うことのできる数値データを直接表記したもので，大きく分

けて，整数型と実数型とがあります．

## 5.3 整数型定数

整数型定数は，表記上の観点から，8進・10進・16進の3つの形式に分類されます．

■10進形式

−32768から＋32767までのすべての整数です．負の整数の場合，数値の前には必ずマイナス符号をつけなければなりませんが，正の数の前の符号は省略できます．また，同じ範囲内の実数の後ろに％をつけると，小数点以下は四捨五入され，整数型の定数とみなされます．

例）　32767
　　　−123
　　　+5
　　　31100.5%　←整数を表す．

■8進形式

頭に&Oまたは&をともなった，0から7までの数字の並び．&0〜&177777の範囲内です．

例）　&12345
　　　&O7777

■16進形式

頭に&Hをともなった，0からFまでの並び．&H0〜&HFFFFの範囲内です．

例）　&H100
　　　&HCFFF

注意：8進形式または16進形式で入力された数値は，PRINTなどの出力文では10進形式で出力されます．10進以外の形式で出力するときは，それぞれOCT＄，HEX＄を使って文字列として出力してください．

## 5.4 実数型定数

実数型定数は，単精度実数型と倍精度実数型に分けられます．

## 5.5 単精度実数型定数

有効桁7桁の精度をもつ実数のデータです．出力のときは7桁目が四捨五入され，6桁以下で表示されます．単精度型実数の値の範囲は，−1.70141E＋38〜1.70141E＋38です．

BASICでは，とくに型宣言をしない場合，数値はみな単精度実数型として扱われます．

単精度実数型の定数には，表記上の分類により，次の3つの形式があります．

■7桁以下の実数

■最後に！をともなった数

■Eを使った指数形式

例）　3525.68

3.14！

−7.09E−06

## 5.6　倍精度実数型定数

有効桁16桁の精度をもつ実数のデータです．出力のときは，16桁以下で表示されます．倍精度型実数の値の範囲は，−1.701411834604692D＋38〜1.701411834604692D＋38です．

倍精度実数型の定数には，表記上の分類により，次の3つの形式があります．

■8桁以上の実数

■最後に#をともなった数

■Dを使った指数形式

例）　1234567.890

56789.0#

−1.09432D−58

# 6.　変数

変数は，プログラム中で使うデータをしまっておくことができる入れ物のようなもので，1バイト系の英数字で名前(変数名)をつけます．変数は，演算，参照などに使うことができます．

なお，変数に値を格納する(入れる)前は，数値変数の値は0，文字変数はヌルストリング(空の文字列)であるものとみなされます．

## 6.1　変数名

変数名は長さ40文字以内の，1バイト系英数字とピリオド(.)の組み合せで表されます．ただし，変数名の初めの1文字は英字でなくてはなりません．この規約にしたがって名前をつけられた変数は，すべて区別されます．

たとえば，次の2つの変数は異なった変数名として解釈されます．

```
COUNTER.OF.TABLE.DATA.999888777666555.01
COUNTER.OF.TABLE.DATA.999888777666555.02
```

変数名は予約語（「第1章　8.　予約語」，「付録E.　予約語一覧」参照）であってはなりませんが，予約語を含んだものはかまいません．ただし，FNで始まる変数名は許されません(第2章中のDEF　FNの項参照)．また英文字において大文字，小文字の区別はありません．

## 6.2 変数の型

変数にも，定数と同様，格納するデータに応じた型があります．

変数に値を格納するためには，代入文やINPUTなどを用いますが，いずれの場合にも変数の型は，格納されるデータの型と一致していなければなりません．

- 変数
  - 文字型
  - 数値型
    - 整数型
    - 実数型
      - 単精度実数型
      - 倍精度実数型

変数の型は，変数名の最後に型宣言文字をつけることによって区別します．型宣言文字の種類は次の4つです．

- 型宣言文字
  - $　文字型
  - %　整数型
  - !　単精度実数型
  - #　倍精度実数型

通常は，型宣言文字を省略すると，〝!〞がついているとみなされます（単精度実数型変数となる）．

例）　A!, A#, A%, A$ ― これらは区別されるが，A!とAは同じ．

ただし，型宣言命令（DEFINT，DEFSNGなど）を用いて一括指定を行った場合，型宣言文字省略時の型は，型宣言命令によって宣言された型となります．たとえば，次の例では，A～Cで始まる変数は，型宣言文字をつけないとみな整数型となります．

例）　DEFINT　A−C

　　　　∫

　　　C=A+B　　　←A，B，Cは，型宣言文字をつけなければ整数型

　　　L=M*N　　　←A～C以外は，型宣言文字をつけなければ単精度実数型

# 7.　配列変数

配列変数は，たくさんのデータを変数に代入したり，参照する場合に使います．たとえば，データの数が100個あるとき，100個の別の変数を用意するのはたいへんです．このようなとき

に，A(0)～A(99) のように変数名の後にカッコつきの番号をつけた形で配列として扱うと，100個のデータを1つの変数名で処理することができます．

カッコ内の番号は添字といい，配列のなかの何番目の要素であるかを表します．たとえば，A(0)は，Aという配列変数の0番目の要素という意味になります．添字には整数定数，または整数型の変数を使うことができます．

配列変数の型(すなわち配列変数内の各要素の型)は，通常の変数と同様，その配列変数名の最後(カッコの直前)につけられた型宣言文字によって決まります．ただし，型宣言命令で一括指定を行った場合は，この限りではありません．詳しくは，「第1章　6.2　変数の型」を参照してください．

プログラム内で配列を使うには，あらかじめDIMによる〝配列の宣言〟が必要です．配列の宣言とは，変数の名前と配列に入れる要素の個数を決めてやることです．

DIMでは次のように，その配列で使用可能な添字の最大値を指定することにより，要素の個数を決めます．

```
DIM  A(5)     添字の最大値が5で，名前が〝A〟という配列変数を宣言
```

添字は，原則として0から始まりますので，実際の要素数は添字の値＋1となります．この例では，5＋1で，要素数は6個となります．

〝添字が0から始まる〟ということを，〝添字の最小値 (OPTION　BASE) が0である〟といい，BASICの起動時にはこの状態になっています．なお，OPTION　BASEという命令を使うことにより，添字の最小値を1にすることもできます．

また，DIM中で配列変数を宣言する際，複数の添字をコンマで区切って指定すると，その配列変数は，添字の個数に応じた〝次元〟数をもつことになります．

たとえば，添字の最小値が0であるとして次のような配列を宣言した場合の，各配列の次元数および要素の合計数を示してみます．

```
DIM  A(10)              1次元配列，要素数は11

DIM  TA(2, 3)           2次元配列，要素数は3×4＝12

DIM  NAME$(2, 5, 3)     3次元配列，要素数は3×6×4＝72
```

2番目の例の場合，次のような2次元(縦方向と横方向の2次元)の配列がとられます．

```
TA(0, 0)  TA(0, 1)  TA(0, 2)  TA(0, 3)
TA(1, 0)  TA(1, 1)  TA(1, 2)  TA(1, 3)
TA(2, 0)  TA(2, 1)  TA(2, 2)  TA(2, 3)
```

配列変数の次元および添字の最大値は，メモリ容量が許す限り，いくつでも設定することができます．ただし，次元については，DIMで1行(255文字)中に記述できる範囲内，という制

限もありますので注意してください．

なお，添字の最大値が10以下（添字の最小値が0でも1でもかまわない）のときは，DIM による宣言を行わなくとも，配列変数を用いることができます．

# 8. 予約語

$N_{88}$-日本語 BASIC(86)(MS-DOS 版)は，命令(関数も含む)の名前，演算子の名前などを，処理上，特殊なものとして扱います．これは〝予約語〟と呼ばれ，文字どおり BASIC によってその使用方法を予約されているものです．したがって，予約語をそのまま変数名として使用したりすることはできません．

予約語を使用する場合には，BASIC がその語を予約語であるか否か認識できるようにするため，語の前，後ろ，またはその両方に，BASIC 文法上で許された，あるいは規定された特殊文字（スペース，ダブルクォーテーション（"），ナンバー記号（#），コロン（：）など）を挿入しなくてはなりません．

BASIC の予約語は，「付録E．予約語一覧」に掲げてあります．

# 9. 型変換

数値データは，必要に応じて他の型に変換することができます．型の変換は次の規則にしたがって行われます．

なお，文字型と数値型の間では型変換を行うことはできませんが，数値表記の文字列に限り，数値との相互変換を関数によって行うことができます(第2章中の STR＄，VAL の項参照)．

(1) ある型の数値データが，違った型の数値変数に代入された場合，数値は，その変数名によって宣言された型に変換されます．

例)
```
10 ABC％=1.234
20 PRINT  ABC％
RUN
  1
```

(2) 精度の違う数値間の演算の場合，精度の高い方に変換されて，演算が行われます．たとえば，10#／3の場合，10#／3#として演算が行われます．

例)
```
10 A#=10#／3
20 B#=10#／3#
30 PRINT  A#, B#
RUN
  3.333333333333333  3.333333333333333
```

(3)　論理演算の場合，扱われる数値はすべて整数に変換され，結果は整数で与えられます．

例）
```
10 A=12.34
20 B=NOT A
30 PRINT B, A
RUN
-13          12.34
```

(4)　実数が整数に変換される場合は，小数点以下は四捨五入されます．このとき，整数型で扱える範囲を超えた場合はエラーが起こります．

例）
```
10 A%=34.4
20 B%=34.5
30 PRINT A%, B%
RUN
 34         35
```
```
10 A#=1.234E+07
20 B%=A#
30 PRINT B%, A#
RUN
Overflow in 20
```

(5)　倍精度実数型変数が単精度実数型変数に代入されたときは，変数の値は有効数字7桁に丸めたものとなります．単精度実数型変数の精度は7桁であり，もとの倍精度の数値との誤差の絶対値は，5.96E－8以下となります．

例）
```
10 A#=1.23456789#
20 B!=A#
30 PRINT A#, B!
RUN
 1.23456789   1.23457
```

倍精度実数型変数(あるいは倍精度実数型定数)と単精度実数型変数(あるいは単精度実数型定数)を混合して演算したり，単精度の値を倍精度実数型変数に代入したりすると，単精度値に変換する際，有効桁以降の桁に変換誤差が混入しますので注意してください．

例1）　精度の異なる数値による演算

●好ましくない例（演算結果に変換誤差が混入する）

```
A#=1.41421356#+0.12
```

●改良例

```
A#=1.41421356#+0.12#
```

例2）　精度の高い変数に対する精度の低い値の代入

●好ましくない例

```
A#=3.1415
```

●改良例

A＃＝3.1415＃

# 10．式と演算

式とは，定数や変数を演算子（計算に使う特殊記号のこと）で結合した一般的な数式をはじめ，たんなる文字列や数値，変数だけのもの，または関数の総称です．たとえば次のものは，すべて式です．

例）　10＋3／5

A＋B／C－D

"BASIC"

3.14

A＄

TAN(D)

演算は，演算子または関数を用いて行う式の操作のことで，次の5つに分類されます．

1．算術演算
2．関係演算
3．論理演算
4．関数
5．文字列演算

以下に，それぞれの演算について説明します．

## 10.1　算術演算

算術演算子には次のようなものがあり，示された順序にもとづいて演算を行います．なお，算術式の中に文字定数や文字変数が入ってはいけません．

| 実行順序 | 算術演算子 | 演算内容 | 例 |
|---|---|---|---|
| ↓ | ^ | 指数(べき乗)演算 | X^Y |
| | － | 負号 | －X |
| | ＊，／ | 乗算，実数の除算 | X＊Y，X／Y |
| | ＋，－ | 加算，減算 | X＋Y，X－Y |

演算の実行順序を変更する場合は，カッコを使用します．カッコの中の演算子は他の演算より先に実行されます．カッコ内においては通常の実行順序に従います．

| | 代数表記 | BASICの表記 |
|---|---|---|
| 1) | $2X+Y$ | 2*X+Y |
| 2) | $\frac{X}{Y}+2$ | X／Y+2 |
| 3) | $\frac{(X+Y)}{2}$ | (X+Y)／2 |
| 4) | $X^2+2X+1$ | X^2+2* X+1 |
| 5) | $X^{Y^2}$ | X^(Y^2) |
| 6) | $(X^Y)^2$ | X^Y^2 |
| 7) | $Y(-X)$ | Y* −X |

なお，演算の実行順序についての詳細は，「第1章　10.6　演算の優先順位」を参照してください.

⑴　**整数の除算と剰余の計算**

整数の除算は¥によって行われます. 扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます.商は小数点以下が切り捨てられた整数となります.

例)　10¥3　→3　　　　(10÷3=3・・・1)
　　23.75¥5　→4　　　(24÷5=4・・・4)

剰余の演算はMODによって行われます. 扱われる数値が実数の場合は，演算が実行される前に小数点以下が四捨五入されます. 結果は整数の割り算の余りです.

例)　13.3　MOD　4　→1　　　　(13÷4=3・・・1)
　　25.68　MOD　6.99　→5　　(26÷7=3・・・5)

⑵　**0での除算**

式の実行時に0での除算が行われた場合は，〝／0″(Division　by　zero)エラーを出力しますが，数値の型に応じてBASICが扱うことのできる最大の数を結果として処理を続行します.

また，0に対して負のべき乗演算を行った場合も同様になります.

例)

```
PRINT  2#／0
／0
1.701411834604692D+38
```

```
PRINT  0^−1
／0
1.70141E+38
```

⑶　**桁あふれ（オーバーフロー）**

代入や演算の結果がその変数の型内で表現することのできる範囲を超えた場合，桁あふれが発生します.

桁あふれが起こった場合，〝OV″(Overflow)エラーを出力し，最大の数を結果として処理を続行します.

例)
```
PRINT  3^300
OV
1.70141E+38
```

## 10.2 関係演算

関係演算子は2つの数値を比較するときに用います．結果は，真(−1)，偽(0)で得られ，条件判定文などプログラムの流れを変えるのに用いられます（第2章中のIF…THEN〜ELSEの項参照）．

| 関係演算子 | 演算内容 | 例 |
|---|---|---|
| = | 等しい | X=Y |
| < >, >< | 等しくない | X< >Y, X><Y |
| < | 小さい | X<Y |
| > | 大きい | X>Y |
| <=, =< | 小さいか等しい | X<=Y, X=<Y |
| >=, => | 大きいか等しい | X>=Y, X=>Y |

**注意**：=は代入文にも使うので注意すること．

IF…THEN〜ELSEの中での使い方の例を次に示します．

例)
```
IF  X=0  THEN  1000
IF  A+B< >0  THEN  X=X+1：Y=Y+1
```

## 10.3 論理演算

論理演算子は，ビット操作やブール演算(2進演算)を行ったり，複数の関係演算式を結合して複合条件を判定したりするのに用いられます．

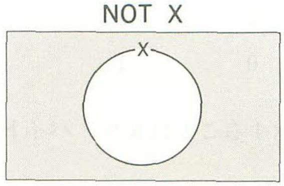
論理演算子は，扱う数値を−32768から+32767までの2の補数表示の整数に変換してから，演算を行います．変換時にこの範囲外となれば〝OV〞(Overflow)エラーとなります．

各論理演算の内容は次のとおりです．論理演算は，扱う整数値の2進表記(16ビット分)に対してビットごとに行われます．次に示すのは，単一のビットX，Yについての演算結果を表しています．

NOT：not（否定）

| X | NOT X |
|---|---|
| 1 | 0 |
| 0 | 1 |

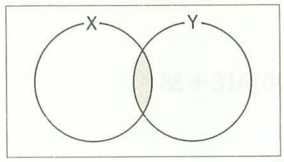
NOT X



AND：and（論理積）

| X | Y | X AND Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

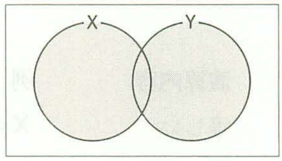
X AND Y




OR：or（論理和）

| X | Y | X OR Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

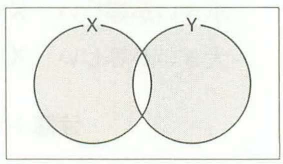
X OR Y




XOR：exclusive or（排他的論理和）

| X | Y | X XOR Y |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

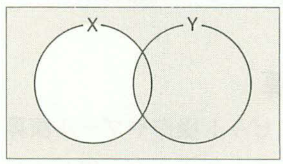
X XOR Y




IMP：implication（包含）

| X | Y | X IMP Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 1 |
| 0 | 0 | 1 |

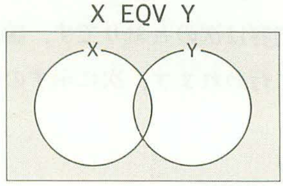
X IMP Y




EQV＝equivalence（同値）

| X | Y | X EQV Y |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

X EQV Y




論理演算を利用することにより，バイトデータをあるビットパターンに照らし合わせて調べることができます．

たとえば，AND演算子は機器の入出力ポートのステータスバイトの必要なビット以外のす

べてのビットをマスクするのに使えます．またOR演算子はある2進数を作るために2つのビットパターンを混合するのに使うことができます．

以下に，論理演算子がどのように働くかの例を示します．演算がビットごとに行われているようすに注意してください．

```
-1 OR 0 →-1              -1=(1111 1111 1111 1111)₂
                           0=(0000 0000 0000 0000)₂
                   -1  OR  0=(1111 1111 1111 1111)₂

48 AND 24 →16             48=(0000 0000 0011 0000)₂
                          24=(0000 0000 0001 1000)₂
                   48 AND 24=(0000 0000 0001 0000)₂

15 XOR 60 →51             15=(0000 0000 0000 1111)₂
                          60=(0000 0000 0011 1100)₂
                   15 XOR 60=(0000 0000 0011 0011)₂

17 EQV 12 →-30            17=(0000 0000 0001 0001)₂
                          12=(0000 0000 0000 1100)₂
                   17 EQV 12=(1111 1111 1110 0010)₂

28 IMP 9 →-21             28=(0000 0000 0001 1100)₂
                           9=(0000 0000 0000 1001)₂
                   28  IMP 9=(1111 1111 1110 1011)₂

NOT 23 →-24               23=(0000 0000 0001 0111)₂
                      NOT 23=(1111 1111 1110 1000)₂
```

論理演算において，もし，0（偽）と−1（真）しか与えられなかったなら，扱う整数値の16ビット全部について同一の演算が行われることになり，演算の結果は0か−1のどちらかとなります（前例の−1　OR　0を参照）．したがって，論理演算子で2つ以上の関係演算を結ぶようにして使用することにより，複合条件によってプログラムの流れを変えるのに用いることができます．

例）IF　X<0　OR　99<X　THEN　1000

　　Xが負または，99より大きければ，行番号1000へ飛ぶ．

IF　0<X　AND　X<100　THEN　X=0

　　Xが正でかつ，100より小さければ，Xに0を代入する．

```
IF NOT (A=0) THEN 20
```

　　Aが0でなければ，行番号20へ飛ぶ．

## 10.4　関数

関数とは，指定されたある値(これを関数の引数(ひきすう)という)に対して，ある決まった演算を行い，その演算の結果を得ることができるものです(ただし，関数によっては，引数を必要としないものもあります)．

関数は，演算子を使用する他の演算とは異なり，実行後にそれ自身が値をもつ，一種の命令として扱われます．したがって，ふつうの命令のように機能別に決められた名前をもっています．

$N_{88}$-日本語BASIC(86)(MS-DOS版)には，SIN（正弦），SQR（平方根）などの数値関数やCHR＄，LEFT＄などの文字列関数といった〝組み込み関数〞が用意されています．これらの関数については，第2章で通常の命令とともに詳しく説明されています．

また，〝ユーザー定義関数〞としてユーザーが自由に定義できる関数機能もあります．これは第2章　DEF　FNの項で説明します．

なお，初等関数（SIN関数などの数学関数）では，引数が整数や単精度のときは単精度の結果が得られますが，引数が倍精度のときは倍精度の結果を得ることができます．

例）
```
A=SIN(3.1416)+COS(3.1416)
PRINT  2, 2*2, SQR(2)
```

## 10.5　文字列の演算

BASICでは，文字列に対して，演算を行うことができます．

**■文字列の連結**

文字列は，演算子“＋”によって連結することができます．

例）
```
10  A$="SUPER":B$="PERSONAL":C$="COMPUTER"
20  D$=A$+"-"+B$+"-"+C$
30  PRINT  D$
RUN
SUPER-PERSONAL-COMPUTER
```

**■文字列の比較**

文字も，数値の比較に用いられるものとまったく同じ関係演算子を用いて比較することができます．

=,　<,　>,　<>,　><,　<=,　=<,　>=,　=>

文字列の場合それぞれの文字列の最初から1文字ずつ，文字の比較を行います．もし相互に

まったく同じ文字列の場合は，その２つの文字列は等しくなりますが，１個所でも違った場合は，その文字のキャラクタコード（「付録C. キャラクタコード表」参照）の大きい方の文字列が大きくなります．文字列の片方が短くて比較が途中で終わった場合は，短い文字列の方が小さくなります．

文字列の比較においては，空白なども意味をもちますから注意してください．

次の例は，すべて真（−1）となります．

例）"AA"<"AB"
　　"BASIC"="BASIC"
　　"X&">"X #"
　　"PEN　">"PEN"
　　"cm">"CM"
　　"DESK"<"DESKS"

文字列の比較は，文字列の内容を調べたり，文字をアルファベット順に並べたり(ソート)するのに使うことができます．

## 10.6　演算の優先順位

種々の演算には優先順位があり，実行時において次の番号順に処理されます．

1　カッコで囲まれた式
2　関数
3　^（指数）
4　−（負号）
5　*, ／（乗算，実数の除算）
6　¥　（整数の除算）
7　MOD（整数の剰余）
8　+, −（加算，減算）
9　関係演算子（<, >, =など）
10　NOT
11　AND
12　OR
13　XOR
14　IMP
15　EQV

# 11.　割り込み

コンピュータの頭脳である CPU (Central Processing Unit) は，一般に同時に２つ以上の処理をすることができません．したがって，１つ仕事を処理している間は他で何が起ころうと，CPU はその処理に移ることはできませんし，それを知ることさえできないのです．このことは，１つの決まった一連の流れを頭から処理するようなプログラムの場合，さほど問題にはなりませんが，１つのプログラムの処理中に，即実行しなければならないような特別なことがら(〝事象〟と呼ぶ)が起きたときに，その処理を先に実行させるようなプログラムを必要とする場合があります．

〝割り込み〟とは，このような処理を実現するための手段です．たとえば，CPUがある処理を実行しているとき，ファンクションキーが押されたなどの特別な事象が発生したとします．このような場合，専用のハードウェアはCPUにその事象が起きたことを，ハードウェア的手段で伝えます．そこでCPUは，現在実行中の処理を一時停止し，特別な事象の処理を優先的に実行し，その処理の終了後，元の処理に復帰するのです．

以上のような流れを，〝割り込み〟と呼ぶわけですが，PC-9800シリーズでは，多くの周辺装置でこの割り込みを使える(すなわち，割り込みの事象を発生させることができる)ようになっており，$N_{88}$-日本語BASIC(86)(MS-DOS版)もこれらの割り込みをサポートしています．

次に，$N_{88}$-日本語BASIC(86)(MS-DOS版)がサポートしている割り込み機能を示します．

(1)　STOPキー　(ON　STOP　GOSUB参照)
(2)　HELPキー　(ON　HELP　GOSUB参照)
(3)　リアルタイム・タイマ　(ON　TIME＄　GOSUB参照)
(4)　ファンクションキー　(ON　KEY　GOSUB参照)
(5)　RS－232C回線　(ON　COM　GOSUB参照)
(6)　マウス　(ON　MOUSE　GOSUB参照)
(7)　サウンド　(ON　PLAY　GOSUB参照)
(8)　GP-IB　(ON　SRQ　GOSUB参照)
(9)　電話制御　(CMD　ON　LINE　GOSUB参照)

## 12.　ラベル名

$N_{88}$-日本語BASIC(86)(MS-DOS版)では，プログラムの分岐先やプログラムの編集時などに利用する行番号の代わりとして，〝ラベル名〟を用いることができます．

あるルーチンの始まりなどに意味のあるラベル名をつけておけば，いちいち行番号を覚えておかなくともよく，またRENUMで行番号のつけ替えを行ってもそのたびに新しい行番号を確かめる必要もなくなるため，プログラムの作成を非常に楽に行うことができます．後でプログラムの修正などを行うときにもたいへん有利です．

まず，ラベル名を使わないで，ふつうに行番号を分岐先として用いたプログラムを例として示します．

```
10 INPUT A
20 IF A<0 THEN 80
30 IF A>0 THEN 60
40 PRINT "zero"
50 GOTO 90
```

```
60  PRINT  "plus"
70  GOTO  90
80  PRINT  "minus"
90  END
```

これは，10行で入力された値が正か負か 0 かを調べるプログラムです．ここで処理の流れを変える命令が20，30，50，70行で使われていて，その分岐先の指定はすべて行番号になっています．

これを，ラベル名を使って書き換えると次のようになります．

```
10  INPUT  A
20  IF  A<0  THEN  *MINUS
30  IF  A>0  THEN  *PLUS
40  PRINT  "zero"
50  GOTO  *EXIT
60  *PLUS : PRINT  "plus"
70  GOTO  *EXIT
80  *MINUS : PRINT  "minus"
90  *EXIT : END
```

プログラムが見やすくなり，処理の内容もよくわかるようになりました．

このように，ラベル名とは，分岐先の目印として自由につけることのできるものです．

ラベル名の使用については，次の注意を守らなければなりません．

(1) ラベル名の頭には必ずアスタリスク(*)をつけなければなりません．

(2) 先頭のアスタリスクを除いて，ラベル名は必ず1バイト系の英文字で始まらなければなりません．

(3) ラベル名に使用できる文字は，先頭のアスタリスクを除いて 1 バイト系英文字と数字とピリオド（.）であり，大文字と小文字の区別はありません．

(4) 予約語をラベルとして使用することはできません．ただし，中に含む場合はかまいません．

(5) ラベル名の長さは，プログラムの 1 行の範囲（255バイト以内）によってのみ制限を受けます．

(6) 参照されるラベル名（呼ばれる方のラベル名）は，必ず行の最初になければなりません．

(7) 1 行中でラベル名の後に続けて命令を記述し，マルチステートメントとするときは，コロン(：)またはスペースによって区切ります．

以上のような制限を無視すると〝Syntax error〞となります．

その他参照されるラベル名に同じものがあった場合，二重定義されているという意味の〝Duplicate label〟エラーが生じます．これらのエラーの検出は，〝RUN〟したとき，プログラムの実行に先だって行われますから，ラベル名のエラーがあるとプログラムを1行も実行することができません．なお，コンパイラの場合，プログラムのコンパイル時にエラーが検出されます．

また，この場合のエラーメッセージはエラー箇所を示す〈行番号〉はともないません．

ラベル名は，プログラム中で分岐の目印として使用できるほか，LIST，DELETEなど〈行番号〉を対象とする命令のパラメータにはすべて使用することが可能です．

例）
```
LIST   *START－*LAST
DELETE   *LOOP1－*EXIT1
```

## 13. 10進文字列とパック10進数

$N_{88}$-日本語BASIC(86)(MS-DOS版)は，通常，数値定数や数値変数(整数型，単精度実数型，倍精度実数型)を内部で2進数データに変換して扱っています．

この場合，入力されたデータ(10進数データ)を内部形式である2進数データに変換する際や，2進数データを出力形式(10進数データ)に変換する際に，変換誤差が生じることがあります．

10進演算機能は，入力データを2進数データに変換せずに10進数データのまま扱うことにより，変換誤差のない演算を行うことができます．

$N_{88}$-日本語BASIC(86)(MS-DOS版)5.0では，10進演算を行うための関数として，PCAL$，PCHK，PCNV，PCNV$の各関数を用意しています．各関数については，第2章をご覧ください．

ここでは，拡張10進数演算機能において取り扱っているデータの形式について説明します．

### 13.1 10進文字列

10進文字列は，文字列定数または文字列変数の一種です．

10進文字列は，数字，ピリオド，スペース，＋，－，および指数表記のためのDまたはEの文字の組み合わせで構成されている文字列です．

例）
```
"123.45"
"0.123 E－10"
```

### 13.2 パック10進数

10進数は，10進文字列そのものではなく，それを圧縮したパック10進数と呼ばれる形式で扱われます．パック10進数は，10バイト固定長の文字列で，9バイト＋1ビットの仮数部と7ビットの指数部から成る実数形式で表されるデータです．パック10進数についての詳細は，$N_{88}$-

日本語 BASIC(86)(MS-DOS 版)5.0 ユーザーズマニュアルを参照してください。

## 14. エラーメッセージ

$N_{88}$-日本語 BASIC(86)(MS-DOS 版)は，プログラムの実行を中断させなければならないようなエラーを実行時に検出したとき，エラーメッセージを画面に出力し，コマンドレベルにもどります。

ダイレクトモードの場合，エラーメッセージは次のような形式で出力されます。

XXXX・・・

プログラムモードの場合は次のように出力されます。

XXXX・・・in　yyyy

XXXX・・・はエラーメッセージで，yyyy はエラーが検出された行番号です。この行はあくまでも，BASIC がエラーを検出した行であり，真のエラーの原因は他の行にある場合がありますのでご注意ください。

エラーメッセージの内容とその対策については，「付録A．エラーメッセージとその対策」を参照してください。

# 第2章

# 命令リファレンス

# 第2章　命令リファレンス

## この章の見方

この章では，$N_{88}$-日本語 BASIC(86)(MS-DOS 版)インタプリタで使用可能なすべての命令(関数も含む)について書式，機能などを解説しています．各命令の解説は次の構成で行われています．

1 → **AKCNV$**　関 数 ← 3　　　2 → C

4 → 機 能　1 バイト系の英数カナ文字を，対応する 2 バイト系全角文字に変換します．

5 → 書 式　AKCNV$(〈文字列〉)

6 → 文 例　A$=AKCNV$(B$)
　　　　　PRINT　AKCNV$("イロハ日本語ABC英語")

7 → 〈文字列〉中の 1 バイト系文字を 2 バイト系全角文字に変換します．〈文字列〉中の 2 バイト系日本語文字に対しては変換操作を行いません．

8 → 注意：この関数は日本語キャラクタモードでのみ使用できます．グラフィックキャラクタモードで使用すると "Illegal function call" エラーとなります．

9 → 参照：KACNV$，サンプルプログラム35

1．**命令の名前．**

2．**命令の種別．** 該当命令をマークで示します．マークの種類と意味は，次のとおりです．

**C**　$N_{88}$-日本語 BASIC(86)コンパイラあるいはコンパイラでコンパイルされたプログラムでその命令が使用できることを意味します．

**G**　GP-IB 拡張機能関連の命令であることを意味します．この命令の使用にあたっては，GP-IB(IEEE-488)インタフェースボードが必要です．

**S**　サウンド拡張機能関連の命令であることを意味します．この命令の使用にあたっては，サウンドボードが必要です．

**N<sub>MS</sub>**　MS-NETWORKS 拡張機能関連の命令であることを意味します．この命令の使用にあたっては，MS-DOS(Ver 3.1)に対する MS-NETWORKS の付加が必要です．

**N**　BRANCH4670 対応ネットワーク拡張機能関連の命令であることを意味します．この命令の使用にあたっては，C&C BRANCH4670 対応 MS-DOS 拡張ソフトウェアによる MS-DOS の機能拡張が必要です．

**P**　モデム-NCU 内蔵電話機制御機能関連の命令であることを意味します.

**D**　10 進演算機能関連の関数であることを意味します.

3. **関数表示.**　該当命令が関数である場合のみ, 〝関数〟と明示してあります.
4. **機能.**　命令の機能を簡単に示します.
5. **書式.**　命令の記述の仕方を示します. 実際の入力時には次のような決まりに従ってください.

- アルファベットの大文字で示された項目は, そのまま1バイト系のアルファベットを入力します. 入力する場合は, 小文字でも大文字でもかまいません. BASIC は入力された命令(行)を, 自動的にすべて大文字に変換します. ただし, ダブルクォーテーション(")で囲まれた文字列や, DATA 行中に記述された文字列などは, 大文字と小文字の区別がなされますから, 必要に応じて使い分けてください.
- カギカッコ（〈　〉）で囲まれた項目は, ユーザーが指定します.
- 角カッコ（〔　〕）で囲まれた項目は, オプションであり省略することができます. 省略した場合, デフォルト値 (BASIC によって設定される値) または以前に指定した値が適用されます.

コンマ（, ）で区切られる複数のパラメータがあり, しかも省略可能なパラメータがある場合, 次の例のような書式で示します.

CONSOLE〔〈スクロール開始行〉〕〔,〈スクロール行数〉〕〔,〈ファンクションキー表示スイッチ〉〕〔,〈カラー／白黒スイッチ〉〕〔,〈キャラクタモードスイッチ〉〕

この場合, 角カッコで囲まれたパラメータを省略することが可能であることを示しています. ここで, あるパラメータ以後をすべて省略する場合は, コンマも含めて省略できます. しかし, 中途のパラメータを省略したうえで後続のパラメータを指定する場合は, それ以前のコンマはすべて指定する必要があります.

たとえば, 上記の例で〈キャラクタモードスイッチ〉のみ指定する場合は, 次のようにします.

```
CONSOLE ,,,, 1
```

なお, 次のようにパラメータの後にコンマまたはセミコロン（; ）を指定するような場合は, コンマならびにセミコロンも含めて省略します.

INPUT　〔〈プロンプト文〉| ; / , |〕〈変数〉〔,〈変数〉…〕

ただし, この約束と異なっている場合もあります. その場合には, 注意書きにてそのことを説明します.

- 前記のカギカッコ，角カッコ以外の記号で丸カッコ ()，コンマ (,)，セミコロン (；)，マイナス記号 (−)，等号 (=) などの記号は示された位置に正しく入力します．
- 省略記号〝…″の続く項目は，1行の許す長さ (255文字) の内で任意の回数繰り返すことができます．たとえば，

  DATA　〈定数〉〔,〈定数〉…〕

  という書式ならば，次のような記述が可能です．

  DATA　0，10，15，20

- 座標指定の所で (Wx，Wy) と記されているのはワールド座標を，(Sx，Sy) はスクリーン座標を表しています．またSTEP (x，y) という記法は相対座標による指定ができることを意味しています．その他たんに (X，Y) と記されているのはキャラクタ座標を表しています．
- 書式の中で〈行番号〉またはそれに類する行番号の指定をさしているものは，行番号の他にラベル名も含んでいると解釈してください．

6. **文例．** 実際の入力の仕方の見本として簡単な例を示します．
7. **解説．** 命令の使用法や詳しい機能を説明します．
8. **注意．** 命令を使ううえで，特に注意すべきことをまとめています．
9. **参照．** 関連の深い他の命令と，サンプルプログラムの番号を示します．

## ABS 関数　C

機能　絶対値を得ます.

書式　ABS(<数式>)

文例　B=ABS(−2)

PRINT　ABS(−1.000000000000001)

〈数式〉に指定した値の絶対値を得ます.

〈数式〉に倍精度実数が含まれる場合, 得られる値は倍精度となりますが, 他の場合には単精度となります.

参照：サンプルプログラム8

## AKCNV$ 関数　C

機能　1バイト系の英数カナ文字を, 対応する2バイト系全角文字に変換します.

書式　AKCNV$(<文字列>)

文例　A$=AKCNV$(B$)

PRINT　AKCNV$("ｲﾛﾊ日本語ABC英語")

〈文字列〉中の1バイト系文字を2バイト系全角文字に変換します. 〈文字列〉中の2バイト系日本語文字に対しては変換操作を行いません.

注意：この関数は日本語キャラクタモードでのみ使用できます. グラフィックキャラクタモードで使用すると〝Illegal function call〞エラーとなります.

参照：KACNV$, サンプルプログラム35

## ASC 関数　C

機能　文字のキャラクタコードを得ます.

書式　ASC(<文字列>)

文例　A=ASC("NEW")

PRINT　ASC(A$)

〈文字列〉中の最初の文字のキャラクタコードを，10進表記で得ます．

参照：CHR$，サンプルプログラム23

## ATN 関数 C

| | |
|---|---|
| 機　能 | 逆正接(アークタンジェント)を得ます． |
| 書　式 | ATN(〈数式〉) |
| 文　例 | ANGLE＝ATN(Y／X)<br>PRINT　ATN(−3.14159／2) |

〈数式〉の値に対する逆正接値を得ます．得られる値はラジアンで，$-\pi/2$から$\pi/2$までの範囲です．

〈数式〉に倍精度実数が含まれる場合，得られる値は倍精度となりますが，他の場合には単精度となります．

参照：COS，SIN，TAN

## ATTR$ 関数 C

| | |
|---|---|
| 機　能 | ファイルの属性を得ます． |
| 書　式 | ATTR$( #〈ファイル番号〉 \| 〈ファイルディスクリプタ〉 ) |
| 文　例 | PRINT　ATTR$(#1)<br>A$＝ATTR$("A：TOOL．BAS") |

オープンされているファイル，ディスク上のファイルの現在の属性文字を得ます．属性文字は以下の意味を持ちます．□は空白を意味します．

"□□"：属性は与えられていません．
"□P"：書き込みが禁止されています．
"E□"：Pオプションつきで SAVE されたプログラムファイルです．

参照：SAVE，SET

# AUTO

| | |
|---|---|
| 機 能 | 行番号を自動的に発生します. |
| 書 式 | AUTO 〔〈行番号〉〕〔,〈増分〉〕 |
| 文 例 | AUTO 100, 5 |

AUTOを実行すると〈行番号〉で指定した行から，以後，リターンキーの入力ごとに〈増分〉ずつ増加した行番号を自動的に発生します.

〈行番号〉,〈増分〉を省略した場合には，ともに10を採用します．コンマがあって〈増分〉を省略した場合には，直前に実行したAUTOコマンドの増分が用いられます.

AUTOを中止してコマンドレベルにもどるには CTRL ＋ C または， STOP キーを押します．このとき，最後に発生した行番号の行は，メモリ中にプログラムとして格納されません.

プログラム中にすでに存在する行と同じ行番号を発生させた場合には，行番号の直後にアスタリスク(*)が表示され，注意を促します．このとき，文字を入力してリターンキーを押すと，その行は新しい内容に入れ代わります．また，何も入力しないでリターンキーを押した場合には，その行は削除されます.

コンパイラにおいてはこの機能は使用できません.

# BEEP

C

| | |
|---|---|
| 機 能 | 内蔵スピーカを鳴らしたり，止めたりします. |
| 書 式 | BEEP 〔〈スイッチ〉〕 |
| 文 例 | BEEP |

〈スイッチ〉の値が1の場合はON(鳴りっぱなし)，0の場合はOFF(鳴りやむ)となります．〈スイッチ〉を省略した場合には，PRINT CHR$(7)を実行したのと同様に，一定時間スピーカを鳴らします.

参照：サンプルプログラム31

# BLOAD C

機　能　機械語ファイルをメモリ上にロードします．

書　式　BLOAD 〈ファイルディスクリプタ〉〔,〈ロードアドレス〉〕〔, R〕

文　例　BLOAD "SUB1", R

BLOAD "SUB2", &H100

〈ファイルディスクリプタ〉には，メモリ上にロードしたい，ディスク上またはRS-232C回線上の機械語ファイル(バイナリイメージのファイル．プログラムでもデータでもかまわない)を指定します．

〈ロードアドレス〉を指定した場合には，直前に実行されたDEF SEGで指定されたセグメントベースに，〈ロードアドレス〉で与えられたアドレス(相対アドレス)を加えた番地からロードし始めます．

〈ロードアドレス〉を省略した場合には，直前に実行されたDEF SEGで指定されたセグメントベースの先頭番地からロードが行われます．

Rオプションを指定すると，機械語ファイルをロード後，直ちにその先頭番地からプログラムとして実行を開始します．したがってこの場合，指定する機械語ファイルは，実行可能なプログラムでなくてはなりません．

Rオプションを指定し，さらに〈ロードアドレス〉をつけてロードする場合には，セーブされた際と異なる番地にロードされても実行可能な性質をもった(リロケータブルな)プログラムを，機械語ファイルとして指定しなければなりません．

なお，Rオプションの指定時には，すでに開かれているデータファイルはその状態を保持します．

機械語プログラム領域のセグメントベースの値は，SEGPTRにより得ることができます．

注意：RS-232C回線上の機械語ファイルをロードする場合，ロードが終了しても自動的にBLOADは終了しません．STOPキーで強制的にブレーク(中止)しなければなりません．

参照：BSAVE, CLEAR, DEF SEG, LOAD, SEGPTR

# BSAVE　C

| 機　能 | メモリ上の指定範囲の内容を，ディスク上あるいはRS-232C回線上に機械語ファイルとしてセーブします． |
|---|---|
| 書　式 | BSAVE　〈ファイルディスクリプタ〉，〈開始アドレス〉，〈長さ〉 |
| 文　例 | BSAVE　"SUB1"，&H100，&H2FF |

〈ファイルディスクリプタ〉には，メモリ上の内容をセーブする際における，ディスク上あるいはRS-232C回線上のファイル名を指定します．

〈開始アドレス〉には，セーブを開始する番地を相対アドレスで指定します．

〈長さ〉には，セーブするメモリ上の内容のバイト数を指定します．

BSAVEは，直前に実行されたDEF　SEGで指定されたセグメントベースに，〈開始アドレス〉の値を加えた番地以降の連続する〈長さ〉バイトの内容を，機械語ファイルとしてセーブします．

機械語プログラム領域のセグメントベースの値は，SEGPTRにより得ることができます．

参照：BLOAD，CLEAR，DEF　SEG，SAVE，SEGPTR

# CALL　C

| 機　能 | メモリ上に用意された機械語サブルーチンを呼び出し，実行します． |
|---|---|
| 書　式 | CALL　〈変数名〉〔(〈引数〉〔，〈引数〉…〕)〕 |
| 文　例 | CALL　MUSBR(X，Y)<br>CALL　M(ARG1$，ARG2$，RESULT$) |

呼び出したい機械語サブルーチンは，あらかじめメモリ上に用意しておかなくてはなりません．このためには，POKEやBLOADを用いることができます．

〈変数名〉には，呼び出したい機械語サブルーチンの実行開始アドレス(相対アドレス)が代入された，変数の名前を指定します．ここで用いる変数には配列変数を用いることはできません．

〈引数〉には，機械語サブルーチンに渡す変数を指定します．引数としてはすべての型の変数を指定することができますが，定数や式の指定はできません．

CALLを実行すると，直前に実行されたDEF　SEGによって指定されたセグメントベースに，〈変数名〉で指定した変数に代入されている相対アドレス値を加えた番地に実行が移ります．CALLによって呼び出されたサブルーチンは，機械語のIRET命令によりBASICに制御をもどすことができます．

〈引数〉の受け渡し方法については，$N_{88}$-日本語BASIC(86)(MS-DOS版)ユーザーズマニュ

アルを参照してください.

参照：BLOAD, BSAVE, CLEAR, DEF SEG, POKE, SEGPTR

# CDBL 関数

C

| 機能 | 整数値，単精度実数値を，倍精度実数値に変換します. |
|---|---|
| 書式 | CDBL(<数式>) |
| 文例 | A#=CDBL(B!／2)<br>A#=CDBL(256%) |

〈数式〉の値を倍精度実数値に変換した値を得ます．ただし，型変換が行われるだけで有効桁数の変化がありませんから，得られた値の精度は，変換する前の型と同じ(整数型なら整数部のみ，単精度実数型なら有効数字6桁)になります.

参照：CINT, CSNG

# CHAIN

C

| 機能 | メモリ上のプログラムからディスク上のプログラムに実行を移します. |
|---|---|
| 書式 | インタプリタ<br>CHAIN 〔MERGE〕〈ファイルディスクリプタ〉〔,〈行番号〉〕〔, ALL〕〔, DELETE〈範囲〉〕<br>コンパイラ<br>CHAIN〈ファイルディスクリプタ〉 |
| 文例 | CHAIN MERGE "TEST.BAS", 1000, ALL, DELETE 500−600 |

■インタプリタの場合

CHAINを実行すると，メモリ上にあるプログラムで使用中のファイルは，そのままの状態でロードされるプログラムに引き継がれます．また，OPTION BASEもロードされるプログラムに引き継がれます.

(1) MERGEオプションの省略時

メモリ上のプログラムは消去され，ディスクから〈ファイルディスクリプタ〉で指定したプログラムがロードされ，実行されます.

〈行番号〉には，ロードされたプログラムの開始行番号を指定します．省略した場合は，プロ

グラムの最初から実行されます．

ALL オプションは，連結後のプログラムに変数，配列を引き渡すのに使います．これを指定した場合すべての変数，配列が引き渡され，省略した場合いっさい引き渡されません．必要な変数，配列だけを引き渡したいという場合は COMMON を使います(COMMON 参照)．

DELETE オプションは，この場合意味をもちません．

**(2) MERGE オプションの指定時**

メモリ上のプログラムと，ディスクからロードされたプログラムとが，連結(マージ)されてひとつのプログラムとなった後，実行されます．

〈ファイルディスクリプタ〉には，ディスクからロードするプログラムを指定します．この場合，ディスクからロードするプログラムは，必ずアスキーセーブされたものでなければなりません．

〈行番号〉には，連結後のプログラムの開始行番号を指定します．行番号を省略した場合には，連結後のプログラムの最初から実行されます．

ALL オプションの扱いは，(1)と同様です．

DELETE オプションおよび〈範囲〉を指定すると，メモリ上のプログラムの指定範囲が削除された後，ディスクのプログラムの連結が行われます．〈範囲〉は，"200－300" のように，行番号をマイナス記号でつないで指定します．DELETE オプションで指定する〈範囲〉の行番号は RENUM を実行すると書き換えられます．また，〈範囲〉の指定には，ラベル名を使うことができます．

**注意**：実行開始行を指定する〈行番号〉は，RENUM を実行しても書き換えられません．また，〈行番号〉としてラベル名を使うことはできません．

DEFINT, DEFSNG, DEFDBL, DEFSTR などの宣言文は，MERGE オプションを指定した場合，連結後のプログラムに対しても有効となりますが，指定がない場合，ロードされたプログラムには何の効果も与えません．

プログラムを連結すると，DEF FN，あるいは ON COM GOSUB, ON ERROR GOTO などの割り込み制御は無効となります．

**■コンパイラの場合**

ロードされるプログラムはメモリ上にあるプログラムと無条件に置き換わります(インタプリタで指定可能な MERGE, ALL, DELETE 等の指定はできません)．必ずロードされるプログラムの先頭に制御が移ります．また，メモリ上にあるプログラム中で使用されていたファイルはそのままの状態でロードされるプログラムに引き継がれます．〈ファイルディスクリプタ〉で指定するプログラムはコンパイラでコンパイルされた EXE 形式のファイルでなければなりません．

変数を引き継ぐ場合，インタプリタと同様COMMONを使用します．

同時オープン可能なファイル数，ランダムファイルのレコードサイズは，個々のプログラムをコンパイルする際プログラム単位に与えられる属性ですが，CHAINで起動されるプログラムではその指定は無効になり，メモリ上にあるプログラムの環境がそのまま起動されるプログラムに引き継がれます．また，OPTION BASEの情報もそのまま引き継がれます．

その他の情報は連結されるプログラムに何の効果も与えません．

**注意**：コンパイラにおいてCHAINを実行した場合，GP-IBの環境は引き継がれません．

参照：COMMON，RUN，サンプルプログラム1，2

# CHDIR　C

**機能**　ディスクのカレントディレクトリを変更します．

**書式**　CHDIR 〔<ドライブ名>〕<ディレクトリ名>

**文例**
```
CHDIR "BIN¥WORK"
CHDIR "B:¥BIN¥WORK"
CHDIR ".."
CHDIR "B:.."
CHDIR "B:..¥TEST"
```

<ドライブ名>で指定されたドライブにあるディスクのカレントディレクトリを，<ディレクトリ名>で指定されたディレクトリに移動します．

<ドライブ名>が省略された場合は，カレントドライブにあるディスクが指定されたものとみなされます．

<ディレクトリ名>には，移動したいディレクトリの名前を，¥で始まる絶対指定か，現在のカレントディレクトリからの相対指定によって指定します．ディレクトリが階層化されている場合には，複数の<ディレクトリ名>を¥でつなぐことにより，目的のディレクトリを指定することもできます．

なお，カレントディレクトリのすぐ上にあるディレクトリを，".."（ピリオド2個）で表すことができます．この表記法は，すぐ上のディレクトリに移動することが極めて多いために用意されているものですが，これを使うことにより，相対指定をより自由に行うことができます．

参照：MKDIR，RMDIR

# CHILD　Ⓒ

| | |
|---|---|
| 機　能 | BASIC から MS-DOS へ一時的に制御を移します. |
| 書　式 | CHILD 〔〈チャイルドプロセス名およびチャイルドプロセスに引き渡す情報〉〕〔, M〕〔, 〈画面クリアスイッチ 1〉〕〔, 〈画面クリアスイッチ 2〉〕〔, 〈ファンクションキー表示スイッチ〉〕 |
| 文　例 | CHILD "DEMOPRO.EXE",1,,1<br>CHILD "DIR B:",M<br>CHILD ,M |

CHILD は, BASIC の動作環境から, 一時的に MS-DOS の動作環境に制御を移し, MS-DOS のプログラムを実行する命令です. このとき, メモリにロードされて MS-DOS 下で実行されるプログラムを, チャイルドプロセスと呼びます.

各パラメータの指定方法を説明します.

**〈チャイルドプロセス名およびチャイルドプロセスに引き渡す情報〉**

このパラメータには, 起動するチャイルドプロセスの名前（プログラム名）とチャイルドプロセスに引き渡す情報を 124 文字以内の文字列で指定します. 基本的な指定形式は, MS-DOS 上でコマンドを実行する場合に入力するコマンド行とまったく同じものです. 例えば, 次のように指定します.

例 1)　"DISKCOPY △A : △B : "　　（△は空白を意味します）

```
"DISKCOPY △A : △B : "
 └───┬───┘ └──┬──┘
     │        引き渡し情報
 チャイルドプロセス名
```

例 2)　"A : SUB1"

```
"A : SUB1"
 └──┬──┘
 チャイルドプロセス名
```

チャイルドプロセス名には, ドライブ名＋プログラムファイル名を指定します. ここで指定するプログラムは, MS-DOS 上で実行可能なファイルでなくてはなりません. ドライブ名が省略されると, カレントドライブが処理の対象となります. 特定のディレクトリ下のプログラムを起動する必要がある場合, ディレクトリ情報（対象となるディレクトリ名）を環境文字列テーブルと呼ばれる領域にあらかじめ設定しておく必要があります（ENVIRON 参照).

引き渡し情報には, チャイルドプロセスに与えるオプション, パラメータ, スイッチなどの任意の文字列を指定しますが, プログラムによっては必要としないものもあります.

〈チャイルドプロセス名およびチャイルドプロセスに引き渡す情報〉を指定した場合, MS-

DOS 下のプログラムの実行が終了すると，制御は自動的に MS-DOS から BASIC へもどります．

〈チャイルドプロセス名およびチャイルドプロセスに引き渡す情報〉は，省略することができます．この場合，MS-DOS に制御が移行したのち，MS-DOS のプロンプトが表示され，コマンド待ちの状態になります．以降は，MS-DOS を直接操作することができます．この場合，BASIC に制御をもどすためには MS-DOS のプロンプトが表示されている状態で EXIT コマンド（MS-DOS の内部コマンド）を実行してください．

M

M は，チャイルドプロセスの実行メモリ環境を設定するものです．次に，M を省略した場合と指定した場合とに分けて説明します．

・M 省略時

M が省略（カンマも含めて省略）された場合には，チャイルドプロセスはメモリのチャイルドプロセス領域にロードされ，実行されます．したがって，この場合，指定されたプログラムをロードするために必要なサイズのチャイルドプロセス領域が，事前に確保されていなければなりません．チャイルドプロセス領域の確保は CLEAR で行います．必要なメモリのサイズは次のようになります．

起動するプログラムそのもののサイズ+18KB（+65KB）
（+65KB は，スタックセグメントが確保されていないプログラムを起動する場合にさらに必要な分）

M を省略してチャイルドプロセス領域にプログラムをロードして実行した場合には，BASIC のいずれの領域も乱されませんので，BASIC プログラム中でチャイルドプロセスを起動した場合でも，チャイルドプロセスの処理終了後，BASIC プログラムの処理を継続することができます．ただし，オープンされていたファイルはクローズされていますので，再オープンが必要です．

・M 指定時

M が指定された場合は，チャイルドプロセスはメモリのシンボルテーブル領域以降にロードされ，実行されます．この場合，チャイルドプロセスをロードするメモリ領域を事前に用意する必要はありません．しかし，チャイルドプロセス実行後，シンボルテーブル以降の領域が乱されますので，BASIC プログラム中からチャイルドプロセスを起動した場合，BASIC プログラムの継続処理は行えません．

インタプリタの場合，チャイルドプロセスから制御がもどってきた直後，BASIC プログラムの処理は中止され，コマンドモードにもどります．このとき，すべての変数／配列の値は初期化されます．なお，BASIC プログラムそのものはメモリに格納されたままですので，再び

BASICプログラムを実行することは可能です.

> コンパイラによりコンパイルされたプログラムの場合，チャイルドプロセスから制御がもどってきた直後，プログラムの実行は自動的に終了します.

**〈画面クリアスイッチ1〉**

BASICからチャイルドプロセス(MS-DOS)に制御を移行するときに，画面をいったん消してから移行するか，BASIC画面をそのまま残したままで移行するかを選択します.

**省略または0**　：　画面がクリアされた後，チャイルドプロセスが起動される.

1　：　画面はそのままの状態でチャイルドプロセスに引き継がれる.

**〈画面クリアスイッチ2〉**

チャイルドプロセス（MS-DOS）からBASICに制御をもどすときに，チャイルドプロセス（MS-DOS）の画面を残したままもどすか，あるいは消してからもどすかを選択します.

**省略または0**　：　チャイルドプロセス（MS-DOS）の画面はそのままBASICに引き継がれる.

1　：　チャイルドプロセス（MS-DOS）の画面は消去される.

**〈ファンクションキー表示スイッチ〉**

チャイルドプロセスを起動した後，MS-DOSのファンクションキーを表示するか否かの選択をします.

**省略または0**　：　MS-DOSのファンクションキーが表示される.

1　：　ファンクションキーは表示されない.

**注意：**(1) CHILDを実行する場合，カレントドライブにセットされているディスクのルートディレクトリ下にコマンドプロセッサ（COMMAND. COM）が用意されていなければなりません. コマンドプロセッサが用意されていない場合，“CHILD　can't execute”のエラーとなります.

(2) チャイルドプロセスに制御が移された後のMS-DOSの画面サイズは，BASICの画面サイズに関係なく無条件に80桁，25行モードになります.

(3) BASICからチャイルドプロセスを起動した直後のカーソル位置は，次のようになります.

BASICの画面を消去後，チャイルドプロセス（MS-DOS）に制御を移行する場合
…ホームポジション

BASICの画面を残して，チャイルドプロセス（MS-DOS）に制御を移行する場合
…最下位行

(4) チャイルドプロセスから制御がもどった直後のカーソル位置は，次のようになります.

チャイルドプロセス（MS-DOS）の画面を残して，BASIC に制御をもどす場合

…最下位行

チャイルドプロセス（MS-DOS）の画面を消去後，BASIC に制御をもどす場合

…ホームポジション

参照：CLEAR，ENVIRON，ENVIRON$

## CHR$ 関数 C

機能　指定したキャラクタコードを持つ文字を得ます．

書式　CHR$(〈数式〉)

文例　A$＝CHR$(65)

PRINT　CHR$(&H4E)

〈数式〉の値のキャラクタコードを持つ文字を得ます．値が 0～255 の範囲にない場合は，"Illegal　function　call" エラーになります．

参照：ASC，JIS$，KNJ$，サンプルプログラム23

## CINT 関数 C

機能　単精度実数値，倍精度実数値を，整数値に変換します．

書式　CINT(〈数式〉)

文例　A%＝CINT(B!＊2)

A%＝CINT(X#)

〈数式〉の値の小数点以下を四捨五入して整数に変換した値を得ます．結果の値が－32768～32767 の範囲にない場合は，"Overflow" エラーになります．

参照：CDBL，CSNG

# CIRCLE

C

機能　円，楕円を描きます．

書式　CIRCLE {(Wx, Wy) | STEP(x, y)}，〈半径〉〔，〈パレット番号 1〉〕〔，〈開始角度〉〕〔，〈終了角度〉〕〔，〈比率〉〕〔，F〔，{〈パレット番号 2〉 | 〈タイルストリング〉}〕〕

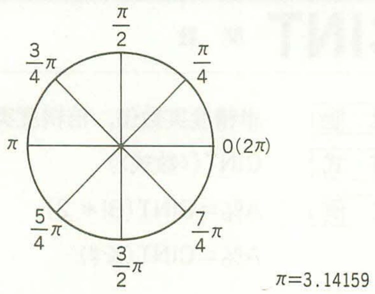
文例　CIRCLE (80, 80), 40, 4, 0, 6.28
CIRCLE STEP(40, −20), 30,,,,, F, TILE$

ワールド座標の(Wx, Wy)を中心とし，〈半径〉で指定される大きさの円を描きます．中心座標は，STEP をつけて LP(最終参照点)からの相対座標により指定することもできます．

〈パレット番号 1〉は，描く円の色をパレット番号で指定します．省略された場合には，COLOR で指定されているフォアグラウンドカラーのパレット番号が用いられます．

〈開始角度〉，〈終了角度〉を指定すると，指定された角度の範囲内のみに円弧を描きます．角度には$-2\pi$〜$2\pi$ の範囲($\pi$ は円周率)のラジアン値を指定します．省略するとそれぞれ，0 と $2\pi$ が採用されます．〈開始角度〉，〈終了角度〉が負であった場合には，その絶対値が用いられますが，そのときには，中心から半径線が描かれますので扇形を描くことができます．

〈比率〉は，円の偏平率を(垂直方向の半径)／(水平方向の半径)で指定します．これを指定することで，楕円を描くことができます．ただし，640×400 ドットのモードでは実際に表示される画面上の長さの比率で指定し，640×200 ドットのモードではその1／2の値を指定します．省略された場合には，640×400 ドットのモードでは 1.0, 640×200 ドットのモードでは 0.5 が用いられます．




〈比率〉が 1 以外(640×400 ドットのとき．640×200 ドットのときは 0.5 以外)の場合，垂直方向の半径と水平方向の半径のうち大きい方に〈半径〉が用いられます．つまり，640×400 ドットのモードの場合，〈半径〉が同じでも〈比率〉が 1 以下ならば横長の楕円となり，1 以上ならば縦長の楕円となります．

F を指定すると，円を描くのと同時にその内部を〈パレット番号 2〉で指定されたパレットの色または〈タイルストリング〉(〔2〕PAINT 参照)により指定された模様でぬりつぶします．〈パレット番号 2〉および〈タイルストリング〉ともに省略された場合には，円を描いたパレットの色でぬりつぶします．開始角度，終了角度が指定されているときに F を指定した場合には，扇

形が描かれ内部がぬりつぶされます.

注意：CIRCLEを実行すると，LP(最終参照点)は円の中心座標(Wx，Wy)に設定されます.

参照：COLOR，〔2〕PAINT，サンプルプログラム14，18，21

# CLEAR

C

機　能　変数の初期化およびメモリレイアウトを決定します.

書　式　インタプリタ

CLEAR 〔〈機械語プログラム領域のサイズ〉〕〔,〈利用者スタック領域のサイズ〉〕〔,〈配列データ領域のサイズ〉〕〔,〈チャイルドプロセス領域のサイズ〉〕

コンパイラ

CLEAR 〔〈機械語プログラム領域のサイズ〉〕〔,〈文字列領域／スタック領域のサイズ〉〕〔,〈配列データ領域のサイズ〉〕〔,〈チャイルドプロセス領域のサイズ〉〕

文　例　CLEAR &H120, &H100

すべての数値変数を0に，また文字変数を""（空の文字列．ヌルストリングともいう）に初期化します．また，DEF(DEF FN，DEF SEG，DEF USR，DEFINTなど)によって定義あるいは指定された情報もすべて無効にします.

CLEARは，このような初期化のほかに，指定されるパラメータに基づいてメモリレイアウトを決定する働きももっています.

まず，BASICインタプリタが動作している場合，およびコンパイラによりコンパイルされたプログラムが動作している場合の，それぞれのメモリレイアウトを示し，各領域の内容を説明します.

## ■インタプリタの場合

**チャイルドプロセス領域**

チャイルドプロセスをロードするための領域（CHILD参照）

CHILDのパラメータにM指定がある場合，この領域は必要としない

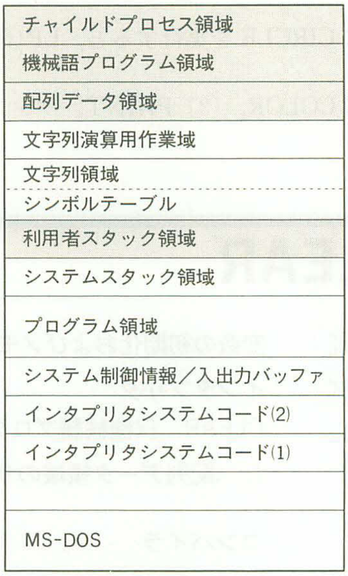
インタプリタ起動直後のサイズ＝0

**機械語プログラム領域**

機械語プログラムをロードするための領域

インタプリタ起動直後のサイズ＝0

インタプリタ




### 配列データ領域

配列制御情報および数値型配列データの格納領域

インタプリタ起動直後のサイズ＝文字列演算用作業域の直後からメモリの上限まで

### シンボルテーブル／文字列領域

変数の制御情報(数値型変数のデータ含む)および文字型変数／文字型配列に代入される文字列データの格納領域

インタプリタ起動直後のサイズ＝最大62KB

### 利用者スタック領域

利用者プログラム実行制御用スタック領域(FOR～NEXT，GOSUB～RETURNなどの制御用)

インタプリタ起動直後のサイズ＝0.5KB

### システムスタック領域

BASICインタプリタの作業用スタック領域

サイズ＝1KB固定

### プログラム領域

BASICプログラムの格納領域

サイズ＝システム制御情報／入出力バッファ，利用者スタック領域およびシステムスタック領域を含めて64KB

### システム制御情報

BASICインタプリタ用各種制御情報の格納領域

### 入出力バッファ

ディスクファイルアクセス用バッファ

サイズ＝BASIC起動時に指定されたファイル同時オープン数＊バッファサイズ

### インタプリタシステムコード (2)

BASICインタプリタ拡張部

### インタプリタシステムコード (1)

BASICインタプリタ本体

## ■コンパイラの場合

| コンパイルされたプログラムの実行時 |
|---|
| チャイルドプロセス領域 |
| 機械語プログラム領域 |
| 配列データ領域 |
| 文字列領域／スタック領域 |
| シンボルテーブル |
| システム制御情報／入出力バッファ |
| P-CODEインタプリタシステムコード(2) |
| 定数領域 |
| P-CODE格納領域 |
| P-CODEインタプリタシステムコード(1) |
| |
| MS-DOS |

### チャイルドプロセス領域

チャイルドプロセスをロードするための領域（CHILD 参照）

CHILD に M パラメータの指定がある場合，この領域は必要としない

プログラム起動時の初期サイズ＝0

### 機械語プログラム領域

機械語プログラムをロードするための領域

プログラム起動時の初期サイズ＝0

### 配列データ領域

配列制御情報および数値型配列データの格納領域

プログラム起動時の初期サイズ＝文字列領域／スタック領域の直後からメモリの上限まで

### 文字列領域／スタック領域

文字型変数／文字型配列に代入される文字列データの格納領域，およびプログラム実行制御用スタック領域

### シンボルテーブル

変数の制御情報(数値型変数のデータ含む)

### システム制御情報／入出力バッファ

P-CODE インタプリタ用各種制御情報の格納領域，およびディスクファイルアクセス用バッファ

プログラム起動時の初期サイズ＝文字列領域／スタック領域，シンボルテーブル，システム制御情報／入出力バッファを合わせて最大 64KB

### P-CODE インタプリタシステムコード（2）

P-CODE インタプリタ拡張部

### 定数領域

プログラム中で使用されている数値定数／文字定数の格納領域

サイズ＝プログラムの構造に依存

### P-CODE プログラム格納領域

コンパイラにより P-CODE 化されたプログラムの格納領域

サイズ＝プログラムの構造（命令形式／命令数）に依存，最大 4 ＊ 64KB

**P-CODE インタプリタシステムコード（1）**

P-CODE インタプリタ本体

これらの領域のうち CLEAR でそのサイズを直接変更できるのは，次の各領域です．

**インタプリタの場合**　：　機械語プログラム領域，利用者スタック領域，配列データ領域，チャイルドプロセス領域

**コンパイラの場合**　：　機械語プログラム領域，文字列領域／スタック領域，配列データ領域，チャイルドプロセス領域

各領域のサイズを変更するためのパラメータの指定のしかたを説明します．

**〈機械語プログラム領域のサイズ〉**

USR や CALL などで呼び出す機械語関数や機械語プログラムを格納するのに必要なサイズを指定します．実際に確保される領域のサイズは，指定した値の 16 倍となります．

機械語関数や機械語プログラムは，SEGPTR (2) で得られるセグメントベースから始まるこの領域内に，BLOAD や POKE を用いてロードするようにします．

機械語プログラム領域は，自動的には確保されませんので，機械語関数や機械語プログラムを使用する場合には，必ず CLEAR を実行するようにしてください．

**〈利用者スタック領域のサイズ〉（インタプリタのみ）**

FOR～NEXT や GOSUB～RETURN などを実行する際に必要となる制御情報を格納しておくためのスタック領域のサイズを指定します．実際に確保される領域のサイズは，指定した値の 16 倍となります．

この領域は，標準で 0.5KB 確保されていますが，FOR～NEXT などの入れ子の多重化によって制御情報が格納しきれなくなると，〝Out of memory〟エラーが発生し，プログラムの実行が停止してしまいます．このような場合，CLEAR によってこの領域を拡大することにより，これを避けることが可能となります．ただし，この領域が拡大した分だけプログラム領域が減少しますので注意してください．

**〈文字列領域／スタック領域〉（コンパイラのみ）**

コンパイラでコンパイルされたプログラムの場合，文字列の格納領域と各種スタック領域とは同一領域内に確保されます．インタプリタの場合と同様に FOR～NEXT などの入れ子の多重化によってプログラムの実行ができないとき，あるいは文字変数への代入量が多すぎてエラーが生じるときには，この領域を拡大することによって問題を解決することができます．実際に確保される領域のサイズは，指定した値の 16 倍となります．

なお，文字列領域／スタック領域，シンボルテーブル，システム制御情報／入出力バッファ

は，合わせて 64KB までしか確保できませんので注意してください.

ちなみに，この領域のサイズは，SEGPTR (4) により得ることができます．また，シンボルテーブル，システム制御情報／入出力バッファを合わせた領域のサイズは，SEGPTR (6) により得ることができます.

〈配列データ領域の大きさ〉

数値配列の値が格納される配列データ領域のサイズを指定します．実際に確保される領域のサイズは，指定した値の 16 倍となります.

DIM の実行時に〝Out of memory〞エラーが発生する場合は，この領域を拡大してください.

〝Out of string space〞エラーの発生する場合は，この領域を縮小することによってシンボルテーブルおよび文字列領域を拡大することができます.

ただしインタプリタの場合，シンボルテーブルおよび文字列領域は 62KB 以上に拡大することはできません．またコンパイラの場合，文字列領域／スタック領域，シンボルテーブル，システム制御情報／入出力バッファを合わせて，64KB 以上に拡大することはできません.

〈チャイルドプロセス領域のサイズ〉

チャイルドプロセスをロードし，実行するための領域のサイズを指定します．実際に確保される領域のサイズは，指定した値の 16 倍となります.

BASIC (P-CODE) プログラムからチャイルドプロセスへ制御を移行し，その後もとのプログラムの実行を継続しようとする場合は，必ずこの領域を確保しなければなりません.

チャイルドプロセスへの制御移行後，もとのプログラムの実行を継続する必要のない場合，すなわちCHILDにMパラメータの指定がある場合は，この領域を確保する必要はありません．この場合，シンボルテーブル以降の領域にチャイルドプロセスがロードされるため，変数，配列，スタックなどの情報はすべてクリアされます．またこの場合，チャイルドプロセスの処理が終了すると，BASIC (P-CODE) プログラムは強制終了されます (P-CODE プログラムの場合，MS-DOS に処理がもどります).

参照 : BLOAD，BSAVE，CHILD，FRE，SEGPTR

# CLOSE　Ⓒ

| 機　能 | ファイルを閉じます. |
|---|---|
| 書　式 | CLOSE　〔〔#〕〈ファイル番号〉〔,〔#〕〈ファイル番号〉〕…〕 |
| 文　例 | CLOSE<br>CLOSE　#1，#3 |

CLOSE は，OPEN によって開かれていたファイルを閉じます．閉じられたファイルに対して

は，再び開かれるまで入出力を行うことはできません．

〈ファイル番号〉を指定すると，OPEN で指定したファイル番号に対応するファイルを閉じます．〈ファイル番号〉を複数個指定すれば，複数のファイルを一度に閉じることができます．〈ファイル番号〉を省略した場合には，そのとき開いているファイルすべてを閉じます．

閉じられたファイルにつけられていたファイル番号は，同じあるいは異なるファイルを開くために再び使うことができます．また閉じられたファイルは，同じあるいは異なったファイル番号によって再び開くことができます．

ファイルが出力用に開かれていた場合には，CLOSE を実行すると，バッファに残っていたデータが完全に書き出されます．このため，ファイルの出力処理を正しく終了するには，CLOSE の実行が必要です．

R 指定なしの RUN および R 指定なしの LOAD は，プログラムの実行を開始する前に，開かれているすべてのファイルを自動的に閉じます．また，END あるいは NEW の実行によってもファイルは自動的に閉じられます．

注意：STOP ではファイルを閉じることはできません．また，END が実行されずにプログラムが終了する場合もファイルは閉じられません．

参照：CHAIN，END，LOAD，NEW，OPEN，STOP，サンプルプログラム26，27，28，29

# CLS

C

| | |
|---|---|
| 機　能 | 現在アクティブな画面をクリアします． |
| 書　式 | CLS　〔〈機能〉〕 |
| 文　例 | CLS　2 |

〈機能〉は 1，2，3 の値をとり，それぞれ次のように働きます．省略された場合には 1 が採用されます．

1　：　テキスト画面のみをクリアします．テキスト表示モードが CONSOLE，〔1〕COLOR により，白黒リバースモードになっている場合には画面は白くなります．

2　：　グラフィック画面のビューポート内をクリアします．グラフィック表示が SCREEN によりカラーモードに設定されている場合には，〔1〕COLOR により指定されたバックグラウンドカラーで，ビューポート内をクリアします．

3　：　テキスト画面，グラフィック画面の両方をクリアします．

テキスト画面をクリアする場合，CONSOLE によって指定されたスクロールウィンドウ内を

クリアします．また，グラフィック画面をクリアした場合には，LP(最終参照点)はビューポートの左上の頂点に移動します．

参照：〔1〕COLOR，CONSOLE，SCREEN，VIEW

# CMD ALLOC　N C

機　能　ディスクサーバ内のボリュームを指定したドライブ番号に接続します．

書　式　CMD　ALLOC　〈ドライブ名〉　AS　〈ボリューム名〉〔；〈パスワード〉〕

文　例　CMD　ALLOC　"C："　AS　"MYDATA；123456"

CMD　ALLOC　"A："　AS　"N88SYSTEM；456789","B："　AS　"MYPROG；000001"

〈ドライブ名〉で指定されたドライブにボリュームを接続します．

〈ドライブ名〉は各仮想ドライブに割り当てられた名前で指定します．

〈ボリューム名〉には，CMD　MKVOL によって作ったボリューム名を１バイト系英数文字16 文字以内で指定します．省略したり""（空の文字列）を入れることはできません．

〈パスワード〉を省略すると"000000"であるとみなされます．

SYSTEM ボリューム，PUBLIC ボリュームはすべてのユーザーが接続することができますが，LOCAL ボリュームおよび WORK ボリュームはオーナーかスーパーユーザーでなければ接続できません．

参照：CMD　FREE，CMD　MKVOL

# CMD BREAK　P

機　能　ブレイク信号の送出を制御します．

書　式　1）CMD　BREAK　〔〔#〕〈電話機番号〉，〕　〈時間〉

2）CMD　BREAK　〔〔#〕〈電話機番号〉〕　ON

3）CMD　BREAK　〔〔#〕〈電話機番号〉〕　OFF

文　例　CMD　BREAK　#1，2

〈電話機番号〉で指定された電話機に，ブレイク信号を送ったり止めたりします．

〈電話機番号〉には，電話機が CMD　LINE　OPEN の実行によって論理的に接続されたときに指定された番号を指定します．〈電話機番号〉を省略した場合は１を指定したものとみなされます．

書式 1）指定された〈時間〉だけブレイク信号を電話機に発信します．〈時間〉の単位は1秒単位で1〜10の値が指定できます．

書式 2）ブレイク信号の発信を開始します．この命令を実行するとCMD　BREAK　OFFを実行するまでブレイク信号を電話機に送出し続けます．

書式 3）ブレイク信号の発信を終了します．CMD　BREAK　ONにより送出されていたブレイク信号を止めます．

注意：この命令はデータ通信モードでのみ可能です．

参照：CMD　LINE　OPEN

# CMD BYE　N C

| | |
|---|---|
| 機　能 | ユーザーの利用終了を宣言します． |
| 書　式 | CMD　BYE |
| 文　例 | CMD　BYE |

ユーザーによる利用を終了し，初期ユーザー名の保護空間に復帰します．

CMD　HELLO　"〈63以下の数字〉"としても同じ動作をします．

参照：CMD　HELLO

# CMD CHANGE DUPLEX　P

| | |
|---|---|
| 機　能 | 通信方式（全二重／半二重方式）を切り換えます． |
| 書　式 | CMD　CHANGE　DUPLEX　〈ポート番号〉〔，〈通信方式〉〕 |
| 文　例 | CMD　CHANGE　DUPLEX　1，1 |

通常，BASICは全二重通信方式でRS-232Cポートを制御します．

ところが，PC-TL101オートホン，PC-9863Nモデムボードあるいは DATAX ITM1200に内蔵されているモデムは，1200ボーの転送速度においては半二重方式による通信しか行えません．

したがって，これらのモデムを介して1200ボーの通信を行う場合，コミュニケーションポートをOPENでオープンする前に，システムの通信方式を半二重方式に切り換えなければなりません．

〈ポート番号〉には通信方式を変更するRS-232Cの回線番号を指定します．

1　:　RS-232C 第 1 回線

2　:　RS-232C 第 2 回線

3　:　RS-232C 第 3 回線

〈通信方式〉に指定する値と通信方式との対応は次のとおりです.

0 または省略　:　全二重

1　:　半二重

**注意**：CMD　CHANGE　DUPLEX は，コミュニケーションファイルのオープンの前に実行しなければなりません．一度，この命令で通信方式を切り換えると，再度この命令で通信方式を切り換えるまで，指定された通信方式が有効となります．

PC-TL102, PC-9865 あるいは ITM1212 を使用する場合は，この命令を実行しないでください．これらの機器においては，全二重方式による 1200 ボーの転送が行われます．

# CMD CONNECT

**機　能**　ワークステーション（ユーザーのコンピュータ）のデバイスをサーバの資源に接続します．

**書　式**　1) CMD　CONNECT　〈ドライブ名〉　AS　〈ネットワーク名〉〔,〈略名〉〕〔,〈パスワード〉〕

2) CMD　CONNECT　〈プリンタファイル名〉　AS　〈ネットワーク名〉〔,〈プリンタ名〉〕〔,〈パスワード〉〕

**文　例**　CMD　CONNECT　"C :"　AS　"SERVER","DISK","USER1"

CMD　CONNECT　"PRN"　AS　"SER","PRINTER"

**書式 1)**

この命令を実行すると，サーバの資源（ディスクやディレクトリ）をワークステーション上のドライブと同様に扱えるようになります．

〈ドライブ名〉には，サーバの資源を接続するドライブを指定します．

〈ネットワーク名〉には，接続する資源を持ったサーバのネットワーク名を，15 文字以内の 1 バイト系英数文字列で指定します．

〈略名〉には，資源につけられた名前を 8 文字以内の 1 バイト系英数文字列で指定します．

〈パスワード〉には，15 文字以内の 1 バイト系英数文字列で指定します．サーバの資源がネットワークにパスワードつきで登録されている場合にだけ必要となります．

書式 2)

この命令を実行すると，LLIST, LPRINT などの出力情報はサーバ側のリモートプリンタに出力されます．

〈プリンタファイル名〉には，ワークステーションのプリンタデバイス（PRN のみ）を指定します．

〈プリンタ名〉には，MS-NETWORKS サーバコマンドでサーバのプリンタにつけられた名前を，8 文字以内の 1 バイト系英数文字列で指定します．〈ネットワーク名〉および〈パスワード〉の指定方法は書式 1) と同様です．

なお，出力情報は，一旦サーバ側のファイルに格納され，SYSTEM の実行，またはプリンタに対する CMD　DISCONNECT の実行を契機にプリンタに出力されます．

参照：CMD　CONT，CMD　DISCONNECT，CMD　PAUSE

# CMD CONT

N<sub>MS</sub> C

**機　能**　CMD　PAUSE により一時的に切り離されたワークステーション（ユーザーのコンピュータ）のデバイスをサーバの資源に再度接続します．

**書　式**　CMD　CONT〔D | P〕

**文　例**　CMD　CONT

CMD　CONT　P

D, P は接続するデバイスを指定するパラメータです．

D　：一時的に切り離したすべてのワークステーションのドライブを，サーバの資源に再度接続する．

P　：一時的に切り離したプリンタデバイスをサーバのプリンタに再度接続する．

省略　：一時的に切り離したすべてのデバイスを，サーバの資源に接続する．

注意：CMD　DISCONNECT によって切り離したデバイスの接続は行えません．

参照：CMD　CONNECT，CMD　DISCONNECT，CMD　PAUSE

# CMD DELIM　Ⓖ Ⓒ

| | |
|---|---|
| 機　能 | デリミタを指定します. |
| 書　式 | CMD　DELIM=<デリミタコード> |
| 文　例 | CMD　DELIM=3 |

この命令は，PRINT@，INPUT@，LINE　INPUT@のデリミタ（区切り記号）を指定します.

<デリミタコード>には 0～3 の数値を指定します．数値とデリミタの対応は次のとおりです.

0　:　$C_R+L_F$

1　:　$C_R$

2　:　$L_F$

3　:　EOI

● EOI を受信した場合，先のデリミタの指定にかかわらず，デリミタを受信したとみなします.

例）トーカ / リスナ

```
CMD  DELIM=3              CMD  DELIM=0
PRINT@1;"ABC"@            INPUT@;A$
```

この場合，バスには"ABC"が送出され，C と同時に EOI が true となります．一方，リスナ側のデリミタは $C_R+L_F$ と指定されていますが，EOI を受信したため，デリミタの受信とみなし，A$に "ABC"を代入します.

●デリミタが EOI である場合，EOI とともに送出されたデータバイトは，データの一部とみなされます.

例）トーカ / リスナ

```
CMD  DELIM=3              CMD  DELIM=3
PRINT@1;"123CR"@          INPUT@;A$
```

この場合，バスには"$123C_R$"が送出され，$C_R$ と同時に EOI が true となります．このとき $C_R$ は，データの一部とみなされるため，A$には"$123C_R$"が代入されます.

●デリミタが EOI である場合，EOI と同時に送出された $C_R$ あるいは $L_F$ は，データの一部とみなされます．デリミタが EOI でない場合は，データの最終の $C_R$，$L_F$ はデータとはみなされません.

例）トーカ / リスナ

```
CMD  DELIM=0              CMD  DELIM=3
PRINT@1;"ABC"@            INPUT@;A$
```

この場合，バスには"ABCC$_R$L$_F$"が送出され，L$_F$ と同時にEOIがtrueになります．このときリスナはEOIをデリミタとみなすため，L$_F$ までをデータとし，A$には"ABCC$_R$L$_F$"が代入されます．

例）トーカ　　　　　　　　　　　リスナ

```
CMD　DELIM=0              CMD　DELIM=2
PRINT@1；"123"            CMD　INPUT@；A$
```

この場合，バスには"123C$_R$L$_F$"が送出されます．リスナはL$_F$ をデリミタとみなしますが，C$_R$ はデータとみなさないため，A$には "123"が代入されます．

- デリミタにEOIを指定した場合でも，PRINT@は@を付加しなければ最後のデータバイトとともにEOIはtrueにはなりません．

# CMD　DIAL　Ⓟ

**機　能**　電話をかけます．

**書　式**　1）CMD　DIAL　〔〔#〕〈電話機番号〉，〕〈電話番号〉〔，〈機能〉〕

　　　　　2）CMD　DIAL　〔〔#〕〈電話機番号〉，〕〈短縮番号〉

**文　例**　CMD　DIAL　#1，TN$，TNC

　　　　　CMD　DIAL　#1，21

書式1）

指定された電話番号の自動発信を行い，指定されたモードの予約を行います．

〈電話機番号〉には，CMD　LINE　OPENで接続された電話機番号を指定します．〈電話機番号〉を省略した場合は1を指定したものとみなされます．

〈電話番号〉は20桁以内の文字列で，次の値を指定できます．

0〜9，＊，#　：　ダイアルコード．通常の電話番号．

：　：　ポーズ．電話機は"："の前の位置までダイアルした後，約2秒程度発信を中断し，その残りの番号をダイアルする．これは，構内交換機（PBX）から外線にダイアルするとき外線への接続要求を行う必要がある場合などに使用する．

（，），－　：　セパレータ．番号を見やすくするために使用．セパレータは特に意味を持たないため，次の電話番号はいずれも同じ動作をする．

034528000

03(452)8000

03－452－8000
(03)(452)(8000)

@　：　再ダイアル．直前にダイアルした電話番号に再度電話をかける．ただし，@と他の文字との同時指定は不可．また，本体の電源をOFFにしたりリセットスイッチを押したりした直後には，@は使用できない．

〈機能〉は自動発信後の動作を決める数値であり，次の数値を指定します．

0または省略　：　ダイアル後通話モード．
相手が話し中の場合や相手が出なし（接続されない）場合には，手動で電話を切る心要がある．相手が出た場合には通話が可能(この場合も，電話を切るのは手動で行う)．

1　：　アンサトーン検出後データ通信モード．
呼び出し相手からアンサートーンが返されるまで待つ．一定時間待ってもアンサートーンが検出されない場合にはエラーとなる．呼び出し相手が出た場合にはデータ通信モードとなり，この命令の実行を終了する．

2　：　接続後データ通信モード．
呼び出し相手が出る（接続される）まで待つ．一定時間待っても相手が出ない場合にはエラーとなる．呼び出し相手が出た場合にはデータ通信モードとなり，この命令の実行を終了する．

3　：　ダイアル後データ通信モード
ダイアル後はすぐ，データ通信モードとなり，この命令の実行を終了する．

〈機能〉に1，2または3を使用する場合，相手にはコンピュータなどの接続された自動着信可能な電話機が接続されていなければなりません．もし，手動着信された場合には，正常に動作しない場合があるので注意が必要です．

〈電話番号〉に再ダイアル"@"を指定し，〈機能〉を省略した場合には，直前に指定した機能が採用されます．機能を指定した場合には，指定した機能が優先されます．

書式2)

〈短縮番号〉で指定された電話番号の自動発信を行います．

〈短縮番号〉は1～40までの電話番号（20個まで）でなければなりません．短縮番号を指定した場合，CMD　STORE　DIALによってその短縮番号に与えられている機能に関係なく，すぐ

次の命令に制御が移行します．なお，この命令はオフラインコマンドモードでのみ使用可能です．

注意：オートダイアルの際には，次のことに注意してください．

(1) 電話機の自動／手動スイッチを〝自動〟にしておきます(PC-9863N, PC-9865モデムボードを使用する場合，この必要はありません)．

(2) 受話器は電話機に置いた状態にしておきます．

(3) 電話機のモードスイッチがAUTOあるいはCALLにセッティングされていなければなりません．

(4) 本体のボーレート指定と電話機の速度スイッチの値が，一致していなければなりません．

(5) PC-9863N，PC-9865モデムボードでは，短縮番号による自動発信は行えません．

参照：CMD　STORE　DIAL

# CMD DISCONNECT

N(MS)　C

機　能　　接続しているワークステーション（ユーザーのコンピュータ）のデバイスとサーバの資源を切り離します．

書　式　　1）CMD　DISCONNECT　〈ドライブ名〉

　　　　　2）CMD　DISCONNECT　〈プリンタファイル名〉

文　例　　CMD　DISCONNECT　"C："

　　　　　CMD　DISCONNECT　"PRN"

**書式1)**

〈ドライブ名〉で指定されたドライブを，それに接続しているサーバの資源（ディスクやディレクトリ）から切り離します．

**書式2)**

〈プリンタファイル名〉で指定されたプリンタデバイスを，それに接続しているサーバのプリンタから切り離します．〈プリンタファイル名〉としてはPRNのみが指定できます．

参照：CMD　CONNECT，CMD　CONT，CMD　PAUSE

# CMD ERASE　[N] [C]

| | |
|---|---|
| 機　能 | ディスクサーバから仮想ボリュームを削除します。 |
| 書　式 | CMD　ERASE　〈ボリューム名〉〔；〈パスワード〉〕〔，〈ボリューム名〉〔；〈パスワード〉〕…〕 |
| 文　例 | CMD　ERASE　"MYDATA；123456","MYPROG；000010" |

ディスクサーバ内に作られた仮想ボリュームを削除します。

〈ボリューム名〉と〈パスワード〉を指定して，削除を行います。

〈パスワード〉を省略すると"000000"とみなされます。それ以外のパラメータ（属性文字，ボリュームタイプ，メモ）は不要です。

複数のボリューム名を，"，"（コンマ）をはさんで並べれば，それらを同時に削除することができます。

一般ユーザーは自分の保護空間内で作ったボリュームを削除することができますが，他の保護空間で作られたボリュームを削除することはできません。SYSTEMボリューム，PUBLICボリュームの削除は，スーパーユーザーかオーナーでなければ行うことはできません。なお，スーパーユーザーはすべてのボリュームを削除することができます。

注意：この命令を実行すると指定された仮想ボリュームは消滅します。仮想ボリューム内に大切なファイルが入っていないか十分確かめてから実行してください。アクティブなボリューム（あるステーションが現在接続しているボリューム）を削除することはできません。

参照：CMD　MKVOL

# CMD ERAUSR　[N] [C]

| | |
|---|---|
| 機　能 | ユーザー名を削除します。 |
| 書　式 | CMD　ERAUSR　〈ユーザー名〉〔，〈ユーザー名〉…〕 |
| 文　例 | CMD　ERAUSR　"NEC001" |

〈ユーザー名〉を手がかりとして識別される保護空間を消去します。

注意：この命令は，スーパーユーザーしか使用することができません。

参照：CMD　MKUSR

# CMD ERROR ON／OFF／STOP P

| 機　能 | 通信エラーによる割り込みを許可，禁止，停止します． |
|---|---|
| 書　式 | 1）CMD　ERROR　〔〔#〕〈電話機番号〉〕　ON |
| | 2）CMD　ERROR　〔〔#〕〈電話機番号〉〕　OFF |
| | 3）CMD　ERROR　〔〔#〕〈電話機番号〉〕　STOP |
| 文　例 | CMD　ERROR　#1　ON |

〈電話機番号〉で指定した電話機の通信エラーによる割り込みを許可，禁止，停止します．なお，ここで述べている通信エラーとは，データ通信モードのときモデム（あるいはモデム電話）からデータを受信した際のエラーを意味しています．

〈電話機番号〉には，CMD　LINE　OPEN で接続された電話機番号を指定します．〈電話機番号〉を省略した場合は1を指定したものとみなされます．

書式 1）割り込み動作を許可します．この命令実行後は，通信エラー割り込みが発生すると，CMD　ON　ERROR　GOSUB によって定義された処理ルーチンへ分岐します．

書式 2）割り込み動作を禁止します．この命令実行後は，通信エラー割り込みが発生しても，処理ルーチンにへの分岐は起こりません．

書式 3）割り込み動作を停止します．この命令実行後は，通信エラー割り込みが発生しても割り込みが起こったことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後 CMD　ERROR　ON によって割り込みが許可されると，その時点で処理ルーチンに分岐します．

参照：CMD　ON　ERROR　GOSUB，サンプルプログラム 40

# CMD FREE N C

| 機　能 | ディスクサーバ内の仮想ボリュームをドライブから切り離します． |
|---|---|
| 書　式 | CMD　FREE　〔〈ドライブ名〉〕〔，〈ドライブ名〉…〕 |
| 文　例 | CMD　FREE　"A："，"B：" |

〈ドライブ名〉で指定したドライブと，そこに接続されていた仮想ボリュームとを切り離します．複数のドライブ名を"，"（コンマ）をはさんで並べれば，複数の仮想ボリュームをいちどに切り離すことができます．

〈ドライブ名〉を省略すると，各ドライブに接続されたボリュームがすべて切り離されます．

参照：CMD　ALLOC

# CMD GET　N C

| 機能 | 自ステーションあての直接メッセージを受信し，ファイルバッファに取り込みます． |
|---|---|
| 書式 | CMD　GET　〈ファイルバッファ番号〉〔，〈変数 1〉〕〔，〈変数 2〉〕〔，｛A｜B｝〕 |
| 文例 | CMD　GET　1，TRADD，MSLEN |

自ステーションあての直接メッセージが着信するまで待ち，着信したらそのメッセージをファイルバッファ内に取り込みます．すでに着信していればすぐにメッセージをファイルバッファに引き取ります．

〈変数 1〉にはメッセージを送信したステーションのステーションアドレス（1～63）が，〈変数 2〉にはメッセージ長（0～256）が，それぞれ自動的に代入されます．

A あるいは B は，受信メッセージ内の日本語コードの認識方法を決定するもので，次のように機能します．

省略　：　コードをシフト JIS コードと認識します．そのまま，ファイルバッファに格納します．

A　：　KI（1A70H）／KO（1A71H）ではさまれたコードを JIS コードと認識します．これらのコードはファイルバッファに格納される際，シフト JIS コードに変換されます．このとき，KI／KO コードは削除されます．

B　：　KI（1B4BH）／KO（1B48H）ではさまれたコードを JIS コードと認識します．これらのコードはファイルバッファに格納される際，シフト JIS コードに変換されます．このとき，KI／KO コードは削除されます．

これらの日本語コード認識方法によって認識されなかったコードは，すべて通常の 1 バイト系英数字・記号・カナとして取り扱われます．

実際にメッセージを受信するためには，あらかじめ FIELD を用いてファイルバッファにフィールド変数を割り付けておきます．そしてメッセージが着信したら，このフィールド変数からメッセージを取り出すようにしてください．

参照：CMD　PUT

# CMD HELLO　N C

| 機　能 | ユーザーの利用開始宣言をします. |
|---|---|
| 書　式 | CMD　HELLO　〈ユーザー名〉 |
| 文　例 | CMD　HELLO　"198375" |

現在の保護空間を，指定した〈ユーザー名〉に対応する保護空間に移行します.

〈ユーザー名〉は6桁以内の64～999999の数字を自由に指定できます.〈ユーザー名〉に63以下の数字を指定すると，CMD　BYEと同じ動作（ユーザーの利用終了の宣言）をします.

ステーションの立ち上げ時には初期ユーザー名でLOGONされた状態になっています.

スーパーユーザー名を指定すればスーパーユーザーになることができます.

**注意**：同一ユーザー名で複数のステーションを同時に使うことはできません.

LOCAL／WORKボリュームをマウントしているとエラーになり，WORKボリュームは消滅します.

初期ユーザー名と同じ名をCMD　HELLOで指定することはできません.

**参照**：CMD　BYE

# CMD LINE CLOSE　P

| 機　能 | BASICと論理的に接続されているモデム-NCU内蔵電話機を切り離します. |
|---|---|
| 書　式 | CMD　LINE　CLOSE〔〔#〕〈電話機番号〉〕 |
| 文　例 | CMD　LINE　CLOSE　#1 |

この命令は電話機とBASICを論理的に切り離す命令で，拡張電話制御命令の使用を終了するものです.切り離し対象となるコミュニケーションポートをOPENでオープンしたファイルがある場合には，この命令の実行により自動的にそのファイルがクローズされます.

〈電話機番号〉には，CMD　LINE　OPENで接続された電話機番号を指定します.また，〈電話機番号〉を省略すると，現在接続されている電話機がすべて切り離されます.

**参照**：CMD　LINE　OPEN

# CMD LINE ON／OFF／STOP　P

機　能　モデム-NCU 内蔵電話機の着信による割り込みを許可，禁止，停止します．

書　式　1）CMD　LINE　〔〔#〕〈電話機番号〉〕　ON

2）CMD　LINE　〔〔#〕〈電話機番号〉〕　OFF

3）CMD　LINE　〔〔#〕〈電話機番号〉〕　STOP

文　例　CMD　LINE　#1　ON

〈電話機番号〉で指定した電話機への着信割り込みを許可，禁止，停止します．

〈電話機番号〉には，CMD　LINE　OPEN で接続された電話機番号を指定します．〈電話機番号〉を省略した場合は1を指定したものとみなされます．

書式1）割り込み動作を許可します．この命令実行後は，着信割り込みが発生すると，CMD　ON　LINE　GOSUB によって定義された処理ルーチンに分岐します．

書式2）割り込み動作を禁止します．この命令実行後は，着信割り込みが発生しても，処理ルーチンへの分岐は起こりません．

書式3）割り込み動作を停止します．この命令実行後は，着信割り込みが発生しても割り込みが起こったことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後 CMD　LINE　ON によって割り込みが許可されると，その時点で処理ルーチンに分岐します．

注意：着信割り込みとは，電話がかかってきたという事象を示しているものであり，CMD　RECEIVE（自動着信）の指定とは無関係に発生します．

STATUS　LINE（関数）を使用しているプログラムでは，正常な着信割り込みが発生しない場合がありますので CMD　LINE　ON／OFF／STOP と混在して使用しないでください．

参照：CMD　LINE　GOSUB，STATUS　LINE，サンプルプログラム 40

# CMD LINE OPEN　P

機　能　モデム-NCU 電話機を BASIC と論理的に接続し，オートダイアル，自動発着信などの機能を利用可能にします．

書　式　CMD　LINE　OPEN　〈ファイルディスクリプタ〉　〔AS〔#〕〈電話機番号〉〕

文　例　CMD　LINE　OPEN　"COM1:"　AS　#1

この命令は電話機をBASICと論理的に接続する命令であり，すべての拡張電話制御命令の実行に先だって実行しなければなりません．

この命令を実行すると，〈ファイルディスクリプタ〉で指定されたコミュニケーションポートにつながっている電話機のモードを，初期モードからオフラインコマンドモードに移行します(電話機のモードの詳細に関しては「$N_{88}$-日本語BASIC(86)(MS-DOS版)ユーザーズマニュアル」を参照)．

〈ファイルディスクリプタ〉には，"COM："("COM1："でも同じ)，"COM2："，"COM3："のいずれかを指定します．ただし，"COM2："，"COM3："は，RS-232C拡張インタフェースボードを実装している場合にのみ指定可能です．

〈電話機番号〉には1～3までの値を指定します．以後，電話制御命令はこの〈電話機番号〉を指定して行います．省略した場合は1が採用されます．

注意：〈電話機番号〉の指定のしかたは，一般のファイル入出力の際の〈ファイル番号〉と似ていますが，〈電話機番号〉と〈ファイル番号〉は別のものですので，同時に同じ番号を使用しても各々が無関係に正常に動作します．

なお，この命令は電話機が初期モードにあるときしか使用できません．

参照：CMD　LINE　CLOSE，STATUS　MODE，サンプルプログラム40

# CMD LOGOFF　N C

機　能　ユーザーの利用終了を宣言します．

書　式　CMD　LOGOFF

文　例　CMD　LOGOFF

現在の保護空間を，初期ユーザー名の保護空間に復帰させます．

参照：CMD　LOGON

# CMD LOGON　N C

機　能　ユーザーの利用開始宣言をします．

書　式　CMD　LOGON　〈ユーザー名〉

文　例　CMD　LOGON　"NEC001"

現在の保護空間を，指定した〈ユーザー名〉に対応する保護空間に移行します．

<ユーザー名>は8文字以内の1バイト系英数字を自由につけることができます。
ステーションの立ち上げ時には初期ユーザー名でLOGONされた状態になっています。
スーパーユーザーのユーザー名を指定すればスーパーユーザーになることができます。

注意：同一ユーザー名で複数のステーションを同時に使うことはできません。LOCAL／WORKボリュームをマウントしているとエラーになり，WORKボリュームは消滅します。
初期ユーザー名と同じ名前をCMD　LOGONで指定することはできません。

参照：CMD　LOGOFF

# CMD LPT CLEAR

N C

| 機　能 | 出力動作中のスプールファイルを削除します。 |
|---|---|
| 書　式 | CMD　LPT　CLEAR |
| 文　例 | CMD　LPT　CLEAR |

CMD　LPT　OPENでオープンしたスプールファイルの内容を削除します。したがってスプールファイルへの出力は中止され，待ち行列にも登録されません。

注意：CMD　LPT　CLEARの実行後は必ずCMD　LPT　CLOSEを行ってください。

参照：CMD　LPT　CLOSE，CMD　LPT　OPEN

# CMD LPT CLOSE

N C

| 機　能 | スプールファイルをクローズし，出力待ち行列に登録します。 |
|---|---|
| 書　式 | CMD　LPT　CLOSE |
| 文　例 | CMD　LPT　CLOSE |

スプールファイルをクローズし，プリンタサーバからローカルプリンタへ出力を切り換えます。同時にローカルプリンタで印字が開始されます。
この命令の実行後は，プリンタ出力関係の命令がローカルプリンタに対して有効になります。

注意：いったん，CMD　LPT　OPENを用いてLPTをオープンしたら，本体の電源を切るまでに必ずCMD　LPT　CLOSEを実行してください。

参照：CMD　LPT　OPEN

# CMD LPT OPEN　N C

機　能　印字出力をローカルプリンタからプリンタサーバに切り換えます．

書　式　CMD　LPT　OPEN　〔WAIT〕〔〈FCB名〉〕

文　例　CMD　LPT　OPEN

CMD　LPT　OPEN　"FCB1"

スプールファイルをオープンし，ローカルプリンタからプリンタサーバに出力を切り換えます．LPRINTやLPRINT　USINGなどの命令がプリンタサーバに対して有効になります．出力データはいったんスプールファイルに転送された後，プリンタサーバに送られます．

〈FCB名〉（システムジェネレーション時に登録してあったFCB名）を指定すると，プリンタサーバのFCB情報による書式制御が行われます．〈FCB名〉を省略したときは規定の書式制御で印刷されます．

WAITパラメータを用いても機能は変わりません．

注意：いったん，CMD　LPT　OPENを用いてLPTをオープンしたら，本体の電源を切るまでに必ずCMD　LPT　CLOSEを実行するようにしてください．

参照：CMD　LPT　CLOSE

# CMD MAIL ON／OFF／STOP　N C

機　能　メッセージの着信による割り込みの許可，禁止，停止を制御します．

書　式　1）CMD　MAIL　ON

2）CMD　MAIL　OFF

3）CMD　MAIL　STOP

文　例　CMD　MAIL　ON

書式1）この命令の次の命令を実行した後，自ステーションあての直接メッセージの着信により，CMD　ON　MAIL　GOSUBによって設定された処理ルーチンに分岐します．ステータスは，〝Enable〟または〝Enable（received）〟に変わります．

MAIL　OFF（Disable）の状態からCMD　MAIL　ONを実行すると、メールバッファはクリアされます．

書式2）自ステーションあての直接メッセージの着信を禁止します．CMD　ON　MAIL　GOSUB使用時には、以後メッセージの着信による割り込み処理ルーチンへの分岐は起こりません．ステータスは〝Disable〟に変わります．

CMD　MAIL　ON の状態から次の操作のいずれかを行うと、CMD　MAIL　OFF を実行したのと同じ状態になります．

RUN
CLEAR
END
編集作業（DELETE を含む）
NEW

書式 3）自ステーションあての直接メッセージの着信による割り込みを停止します．以後割り込みを覚えているだけで処理ルーチンへの分岐は起こりません．その後，CMD　MAIL　ON によって割り込みが許可されると，前の割り込みにより処理ルーチンに分岐します．ステータスは〝Pending〟または〝Pending(receive)〟に変わります．〝Pending〟の状態であれば，メッセージを受信できます．

参照：CMD　ON　MAIL　GOSUB，CMD　STATUS／LSTATUS

# CMD　MDSUSR　N C

| 機　能 | スーパーユーザー名の変更をします． |
|---|---|
| 書　式 | CMD　MDSUSR　〈ユーザー名〉 |
| 文　例 | CMD　MDSUSR　"NEC001" |

スーパーユーザー名を，〈ユーザー名〉で指定された名前に変更します．

注意：この命令は，スーパーユーザーしか使用することができません．

# CMD　MKUSR　N C

| 機　能 | ユーザー名の登録をします． |
|---|---|
| 書　式 | CMD　MKUSR　〈ユーザー名〉〔，〈ユーザー名〉…〕 |
| 文　例 | CMD　MKUSR　"NEC001" |

〈ユーザー名〉で指定されたユーザー名に対応する保護空間を，新たに登録します．

注意：この命令は，スーパーユーザーしか使用することができません．

参照：CMD　ERAUSR

# CMD MKVOL　N C

**機能**　ディスクサーバ内に新規にボリュームを作成します.

**書式**　CMD MKVOL 〈ボリューム名〉〔;〈パスワード〉〕〔,〈属性文字〉〕〔,〈ボリュームタイプ〉〕〔,〈メモ〉〕

**文例**　CMD MKVOL "MYDATA;123456","W",,"D"

〈ボリューム名〉は16文字以内の文字列で，自由に定義することができます．ただし，コントロールコード（20H未満），"*"および"?"は使うことができません．また，""（空の文字列）およびすべてがスペースのみの文字列も使うことはできません．

〈パスワード〉は，〈ボリューム名〉の後にセミコロン（;）で区切って6桁の数字で指定します．省略すると，"000000"がパスワードとして設定されます．

〈属性文字〉は，ボリュームの作成される保護空間の属性を指定します．〈属性文字〉に指定する値とそのときに与えられる属性は次のとおりです．

| 〈属性文字〉 | | 属性 |
|---|---|---|
| S | : | SYSTEM |
| P | : | PUBLIC |
| Lまたは省略 | : | LOCAL |
| W | : | WORK |

〈ボリュームタイプ〉は，ファイルサーバ内にどのタイプのディスクに相当するボリュームを作成するかを指定します．〈ボリュームタイプ〉に指定する値とボリュームタイプ，最大許容サイズとの関係は次のとおりです．

| 〈ボリュームタイプ〉 | | ボリュームタイプ | 最大許容サイズ |
|---|---|---|---|
| 1 | : | 160KBタイプフロッピィディスク | 35トラック |
| 2 | : | 320KBタイプフロッピィディスク | 80トラック |
| 3または省略 | : | 640KBタイプフロッピィディスク | 160トラック |
| S | : | 1MBタイプフロッピィディスク | 154トラック |
| E | : | 5インチ固定ディスク（5MB） | 5MB |
| F | : | 5インチ固定ディスク（10MB） | 10MB |
| G | : | 5インチ固定ディスク（20MB） | 20MB |

〈メモ〉はユーザーが自由に1文字（キャラクタ，数字）を指定することができます．

**注意**：SYSTEM属性またはPUBLIC属性で用いるボリューム名と，あるユーザーの保護空間においてLOCAL属性またはWORK属性で用いるボリューム名とは異なっていなくて

はなりません．もし同じであると，エラーとなります．

〈属性文字〉，〈ボリュームタイプ〉を省略する場合，コンマ（,）を省略することはできません．その場合，文例のようにコンマだけを指定します．

参照：CMD　ERASE，CMD　VOLS

# CMD MODE CUT　P

| 機　能 | 電話を切ります． |
|---|---|
| 書　式 | CMD　MODE　CUT　〔(#)〈電話機番号〉〕 |
| 文　例 | CMD　MODE　CUT　#1 |

〈電話機番号〉で指定された電話を切って，オフラインコマンドモードにもどります．〈電話機番号〉を省略した場合は1を指定したものとみなされます．

注意：この命令はデータ通信モードで有効です．

# CMD MODIFY　N C

| 機　能 | ボリューム名，属性，タイプ，メモを変更します． |
|---|---|
| 書　式 | CMD　MODIFY　〈旧ボリューム名〉〔；〈旧パスワード〉〕 AS 〔〈新ボリューム名〉〔；〈新パスワード〉〕〕〔，〈新属性文字〉〕〔，〈新ボリュームタイプ〉〕〔，〈新メモ〉〕 |
| 文　例 | CMD　MODIFY　"MYDATA；123456"　AS,"W",,"D" |

〈旧ボリューム名〉〔；〈旧パスワード〉〕で指定される仮想ボリュームのボリューム名，パスワード，属性，タイプ，メモを〈新ボリューム名〉〔；〈新パスワード〉〕，〈新属性文字〉，〈新ボリュームタイプ〉，〈新メモ〉で指定したものに変更します．

一般ユーザーは自分のユーザー名に対応する保護空間内で作ったボリューム名，属性，タイプ，メモを変更することができますが，他の保護空間で作られたボリュームは変更することができません．

SYSTEMボリューム，PUBLICボリュームについての変更はスーパーユーザーかオーナーでなければ行えません（スーパーユーザーはすべてのボリュームを変更することができます）．

なお，アクティブ（あるステーションが現在接続しているボリューム）なボリュームのボリューム名などを変更することはできません．

**注意**：ファイルのセーブされているボリュームのボリュームタイプを変更しないでください．
読み出せなくなります．
パスワードを省略すると"000000"とみなされます．

## CMD ON ERROR GOSUB Ⓟ

| | |
|---|---|
| 機能 | 通信エラーによる割り込みルーチンの開始行を定義します． |
| 書式 | CMD ON ERROR 〔〔#〕<電話機番号>〕 GOSUB <行番号> |
| 文例 | CMD ON ERROR #1 GOSUB *TEL.REC |

<電話機番号>で指定した電話機に通信エラー割り込みが発生したときに分岐する，処理ルーチンの開始行を定義します．

<電話機番号>には，CMD LINE OPEN で接続された電話機番号を指定します．<電話機番号> を省略した場合は1を指定したものとみなされます．

<行番号> には処理ルーチンの開始行番号を指定します．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURN によって行います．たんに RETURN としたときは中断したところから再開し，後ろに行番号を指定したときはその行から実行を再開します．

ここで述べている通信エラーとは，データ通信モードのときモデム（またはモデム電話）からデータを受信した際のエラーを意味しています．

この命令によって割り込み処理ルーチンへ分岐させるには，この事象に対する割り込みが許可状態（CMD ERROR ON の状態）になっていなくてはなりません．

**注意**：通信エラーの情報は，コミュニケーションバッファ内に記憶されていますので，割り込み処理ルーチン内で INPUT$(関数)を使用してバッファ内のデータを読み取らないと，エラー情報はリセットされません．

**参照**：CMD ERROR ON／OFF／STOP，サンプルプログラム 40

## CMD ON LINE GOSUB Ⓟ

| | |
|---|---|
| 機能 | モデム-NCU 内蔵電話機の着信による割り込みが発生したときの処理ルーチンの開始行を定義します． |
| 書式 | CMD ON LINE 〔〔#〕<電話機番号>〕 GOSUB <行番号> |
| 文例 | CMD ON LINE #1 GOSUB *TEL.REC |

〈電話機番号〉で指定した電話機に着信割り込みがあったときに分岐する，処理ルーチンの開始行を定義します．

〈電話機番号〉には，CMD　LINE　OPEN で接続された電話機番号を指定します．〈電話機番号〉を省略した場合は 1 を指定したものとみなされます．

〈行番号〉には処理ルーチンの開始行を指定します．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURN によって行います．たんに RETURN としたときは中断したところから再開し，後ろに行番号を指定したときは，その行から実行を再開します．

この命令によって割り込み処理ルーチンへ分岐させるには，この事象に対する割り込みが許可状態（CMD　LINE　ON の状態）になっていなくてはなりません．

**注意**：着信割り込みとは，電話がかかってきたという事象を示すもので，CMD　RECEIVE（自動着信）の指定とは無関係に発生します．
STATUS　LINE（関数）を使用しているプログラムでは，正常に着信割り込みが発生しない場合がありますので，CMD　ON　LINE　GOSUB と混在して使用しないでください．

**参照**：STATUS　LINE，サンプルプログラム 40

# CMD ON MAIL GOSUB　N C

| | |
|---|---|
| 機　能 | メッセージの着信による割り込み処理ルーチンの開始行を定義します． |
| 書　式 | CMD　ON　MAIL　GOSUB　〈行番号〉 |
| 文　例 | CMD　ON　MAIL　GOSUB　＊LABEL |

自ステーションあてメッセージを受信すると，指定された〈行番号〉から始まる処理ルーチンに分岐します．

この割り込み指定は 1 メッセージにつき 1 回動作します．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURN によって行います．たんに RETURN としたときは中断したところから再開し，後ろに行番号を指定したときは，その行から実行を再開します．

この命令によって割り込み処理ルーチンへ分岐させるには，この事象に対する割り込みが許可状態（CMD　MAIL　ON の状態）になっていなくてはなりません．

**注意**：INPUT の入力待ちのときは着信割り込みはかかりません．
メッセージ着信による割り込みは，他の $N_{88}$-日本語 BASIC(86)(MS-DOS 版)の割り込みとは異なり，割り込み処理ルーチンに制御が移っても自動的に割り込み停止状態とは

なりません．ですから，割り込み処理ルーチンの先頭でCMD　MAIL　STOPを実行し，処理ルーチンの終わりでCMD　MAIL　ONを実行する必要があります．

参照：CMD　MAIL　ON／OFF／STOP

# CMD PAUSE

N<sub>MS</sub> C

機　能　ワークステーション（ユーザーのコンピュータ）のデバイスをサーバの資源から一時的に切り離します．

書　式　CMD　PAUSE〔 D ｜ P 〕

文　例　CMD　PAUSE

CMD　PAUSE　P

D，Pは切り離すデバイスを指定するパラメータです．

D　：　サーバの資源に接続しているワークステーションのすべてのドライブを一時的に切り離す．

P　：　接続しているサーバのプリンタを一時的に切り離す．

省略　：　すべてのデバイスを，サーバの資源から一時的に切り離す．

参照：CMD　CONNECT，CMD　CONT，CMD　DISCONNECT

# CMD PPR

G C

機　能　PPRモードを指定します．

書　式　CMD　PPR=〈モード〉

文　例　CMD　PPR=1

GP-IBコントローラからのパラレルポールに対する応答（PPR）のモードを指定します．〈モード〉に指定する値とモードの関係は次のとおりです．

0　：　パラレルポートに応答しない．

1　：　パラレルポートに応答する．

2　：　SRQ（サービスリクエスト）送信時，パラレルポールに応答する．

この命令は，スレーブモードでのみ使用できます．

● μPD7210がすでに，データを受信している状態で，インタフェースバスをコマンドモードにした場合，受信済みのデータは失われます．

例）
```
10 FOR I=1 TO 10
20 INPUT@1 ; A$(I)
30 NEXT
```

このプログラムの20行で，トーカ1から送信されてきたデータを受信後，次に20行を実行するまでにデータが送信されてきていると，そのデータの先頭1バイトは失われます．

このような場合，次のようにプログラムを変えてください．

```
5  INPUT@1 ; A$(1)
10 FOR I=2 TO 10
20 INPUT@ ; A$(I)
30 NEXT
```

参照：INPUT@

# CMD PUT

N C

**機能**　他のステーションへ直接メッセージを送ります．

**書式**　CMD PUT 〈ファイルバッファ番号〉,〈受信者アドレス〉〔,〈メッセージ長〉〕〔, | A | 〕
| B |

**文例**　CMD PUT 1, 3, 128

指定したステーションへ，ファイルサーバを経由せずに直接メッセージを送ります．

〈ファイルバッファ番号〉はFIELDで指定したバッファ番号です．

〈受信者アドレス〉はメッセージの送信先ステーションのステーションアドレス（1〜63）を指定します．255を指定すると全ステーションにメッセージが送られます．

〈メッセージ長〉は，0以上256以下でメッセージの長さを指定します（単位：バイト）．省略すると256が採用されます．

受信側のステーションが着信許可の状態にないときは〝Line error〟となります．

送信側のステーションはメールの着信許可／禁止／保留に関係なく送信できます．

AあるいはBは，次のように日本語コードの送出方式を決定します．

省略　：　文字データをシフトJISコードと認識し，そのまま送出する．

A　：　シフトJISコードをJISコードに変換し，前後にKI(1A70H)／KO(1A71H)を付加して送出する．

B　：シフトJISコードをJISコードに変換し，前後にKI(1B4BH)／KO(1B48H)を付加して送出する．

実際にメッセージを送るには，あらかじめFIELDを用いてファイルバッファにフィールド変数を割り付け，送出したいメッセージをこのフィールド変数にセット（LSET／RSET）してからCMD　PUTを実行してください．

注意：AあるいはBを指定した場合，JISコードの前後にKI／KOコードが付加されますが，〈メッセージ長〉には付加されるKI／KOコードを含めた長さを指定しなければなりません．

参照：CMD　GET

# CMD RECEIVE　Ⓟ

| 機　能 | 電話機の自動着信を行うか否かの設定をします． |
|---|---|
| 書　式 | CMD　RECEIVE　〔〔#〕〈電話機番号〉,〕〔〈機能〉〕 |
| 文　例 | CMD　RECEIVE　#1<br>CMD　RECEIVE　1 |

〈電話機番号〉には，CMD　LINE　OPENで接続された電話機番号を指定します．〈電話機番号〉を省略した場合は1を指定したものとみなされます．

〈機能〉には，自動着信後の動作モードを規定する数値を指定します．指定する数値と動作モードとの関係は次のとおりです．

0または省略　：　自動着信を行わない．電話がかかってきた場合，電話機のベルが鳴り受話器を取るのを待つ．

1　：　着信後，データ通信モードに切り換える．電話がかかってきた場合，アンサートーンを返した後にデータ通信モードとなる．
着信割り込み処理ルーチン内でこの命令を実行すると，ただちにアンサートーンを返してデータ通信モードとなる．

注意：着信を検出してからデータ通信モードに移行するまでには時間がかかりますので，STATUS　MODE（関数）を使用してデータ通信モードに移行したことを確認した後，データ通信を行ってください．
この命令はオフラインコマンドモードでのみ有効です．

参照：STATUS　MODE，サンプルプログラム40

# CMD RETRACT

N C

| | |
|---|---|
| 機　能 | PC　NETとの互換用で，実際にはなにも実行しません． |
| 書　式 | CMD　RETRACT |
| 文　例 | CMD　RETRACT |

無効な命令です．

# CMD RETURN

N C

| | |
|---|---|
| 機　能 | 初期ボリュームの接続状態にもどしてステーションを再起動します． |
| 書　式 | CMD　RETURN |
| 文　例 | CMD　RETURN |

現在接続されているボリュームを切り離した後，初期ボリュームの接続状態（イニシャルアロケーションテーブル）にもどり，ステーションを再起動します．

参照：CMD　START

# CMD SERVER

N C

| | |
|---|---|
| 機　能 | ネットワーク上にある複数のデバイスサーバを切り換えます． |
| 書　式 | CMD　SERVER　〈サーバアドレス〉〔，X〕〔，N〕 |
| 文　例 | CMD　SERVER　2,，N |
| | CMD　SERVER　2 |

〈サーバアドレス〉で指定されたサーバにデバイスサーバを切り換えます（システム起動時のサーバは0です）．

Xを指定すると，切り換えられたサーバ上で20MBの固定ディスクタイプボリュームをアクセスすることが可能です．システム起動時は10MBの固定ディスクタイプボリュームまでしかアクセスできない状態になっていますので，カレントサーバ上で20MBの固定ディスクボリュームをアクセスしたい場合，CMD　SERVERでXを指定してください．

ただし，サーバ側が対応していない場合はXは指定しないでください．

Nを指定するとボリュームの接続状態についての情報が保持されるので，再びもとのサーバにもどってきたとき，前と同じボリュームの接続状態が実現されます．Nを省略すると，それ

まで接続されていたボリュームはすべて切り離され，オープンされていたファイルは自動的にクローズされます．

ステーションをリセットしてもサーバアドレスは変わりませんが，電源を切るか，CMD　SERVER　0を実行するともとにもどります．

# CMD START　N C

機能　ボリュームとドライブとを接続した後，IPL動作を実行します．

書式　CMD　START　〈ドライブ名〉　AS　〈ボリューム名〉〔；〈パスワード〉〕〔，〈ドライブ名〉　AS　〈ボリューム名〉〔；〈パスワード〉〕…〕

文例　CMD　START　"A："　AS　"N88SYSTEM；123456"，"B："　AS　"DATA"

ボリュームとドライブとを指定に従って接続した後，ステーションを再起動します．

〈ドライブ名〉は仮想ドライブに割り当てられた名前を指定します．

〈ボリューム名〉はディスクサーバ内の仮想ボリュームの名前です．仮想ボリュームにパスワードが設定されていれば，〈パスワード〉に正しいパスワードを指定するようにします．

コンマ（，）で区切って指定することにより，複数のドライブとボリュームの接続を行うことができます．

注意：CMD　STARTでは，仮想ドライブ中でいちばん小さい番号を持つドライブに接続するボリュームは，システムボリュームでなければなりません．そうでないときには以後の動作は保証されませんので注意してください．

パスワードは省略すると"000000"とみなされます．

参照：CMD　RETURN

# CMD STATE／CMD LSTATE　N<sub>MS</sub> C

機能　ワークステーションのデバイス名とそれに接続しているサーバの資源をディスプレイやプリンタに出力します．

書式　1）CMD　STATE

2）CMD　LSTATE

文例　CMD　STATE

CMD　LSTATE

ワークステーション（ユーザーのコンピュータ）のデバイスとサーバの資源（サーバのネットワーク名，資源の略名など）との接続状況を，書式1）はディスプレイ画面に，書式2）はプリンタに出力します．

## CMD STATUS／CMD LSTATUS　N C

機　能　各仮想ドライブに接続されているボリューム名などを，ディスプレイ画面やプリンタに出力します．

書　式　1）CMD　STATUS

2）CMD　LSTATUS

文　例　CMD　STATUS

CMD　LSTATUS

各ドライブにボリュームが接続されているかどうかを，書式1）はディスプレイ画面に，書式2）はプリンタに表示します．

ボリュームが接続されている場合はそのボリューム名などのほか，メッセージ受信の許可／禁止／保留の状況，プリンタの出力先（ローカルプリンタまたはプリンタサーバ）が表示されます．

仮想ボリュームが接続されていないドライブは〝Not　Active〞と表示されます．

**注意**：ローカルディスクが接続されている場合は，ドライブ番号とタイプのみ表示されます．

CMD　MAIL　STOPを実行すると，Mailの欄は〝Pending〞と表示されます．

## CMD STOP SERVER　N C

機　能　デバイスサーバを終了させます．

書　式　CMD　STOP　SERVER

文　例　CMD　STOP　SERVER

デバイスサーバを終了させます．このコマンド実行後はメールを除きネットワークを利用できなくなります．利用しようとすると〝Not　service〞のエラーになります．

**注意**：この命令は，スーパーユーザーしか使用することができません．

# CMD STORE DIAL

P

| | |
|---|---|
| 機 能 | 短縮ダイアルを電話機に記憶させます. |
| 書 式 | CMD STORE DIAL 〔〔#〕〈電話機番号〉,〕〈短縮番号〉 AS 〈電話番号〉〔,〈機能〉〕 |
| 文 例 | CMD STORE DIAL #1, 21 AS "03(117)1111" |

電話番号を電話機に記憶させる命令です.

〈電話機番号〉には，CMD LINE OPEN で接続された〈電話機番号〉を指定します.〈電話機番号〉を省略した場合は1を指定したものとみなされます.

〈短縮番号〉には，1〜40の範囲の数値を指定します．全部で合計20個までの番号を登録することができます.

〈電話番号〉には記憶させる電話番号を20桁以内の文字列で指定します．指定できる値は次のとおりです.

| | | |
|---|---|---|
| 0〜9, *, # | : | ダイアルコード |
| : | : | ポーズ |
| ( , ) , − | : | セパレータ |

また,〈電話番号〉に""(空の文字列)を指定すると，その短縮番号に記憶されている内容がクリアされます.

〈機能〉には自動発信後の動作を規定する数値を指定します.

| | | |
|---|---|---|
| 0または省略 | : | ダイアル後通話モードとなる. |
| 1 | : | アンサートーン検出後データ通信モードとなる. |
| 2 | : | 接続後データ通信モードとなる. |
| 3 | : | ダイアル後データ通信モードとなる. |

〈電話番号〉および〈機能〉の詳細は，CMD DIAL を参照してください.

**注意**：この命令はオフラインコマンドモードでのみ使用可能です．PC-9863N，PC-9865 モデムボードでは，この命令を使用することはできません.

**参照**：CMD DIAL

# CMD TIMEOUT　G C

機　能　タイムアウトチェックのリミット値を指定します.

書　式　CMD　TIMEOUT＝〈タイムアウト時間〉

文　例　CMD　TIMEOUT＝10

タイムアウトチェックは，ハンドシェークなどのバス機能がハングアップした場合に行われます.

〈タイムアウト時間〉には，0〜255の数値でタイムチェックの最大待ち時間を指定します．単位は秒ですが，正確ではありません．IEEE-488ステートメントの実行が開始されると自動的にタイムアウトチェックが始まり，指定された時間が経過してもその状態から抜け出せない場合は，バスエラーとして処理されます.

〈タイムアウト時間〉を0にすると，タイムアウトチェックは行われません.

# CMD VOLS／CMD LVOLS　N C

機　能　現在の保護空間内にある仮想ボリュームの一覧表を，CRT画面あるいはプリンタに出力します.

書　式　1) CMD　VOLS　〔〈モードバイト〉〕〔,〈ボリュームタイプ〉〕〔,〈メモ〉〕〔,〈パスワード〉〕〔,〈ボリューム名〉〕

2) CMD　LVOLS　〔〈モードバイト〉〕〔,〈ボリュームタイプ〉〕〔,〈メモ〉〕〔,〈パスワード〉〕〔,〈ボリューム名〉〕

文　例　CMD　VOLS　L, G, , ,"TEST???"

〈モードバイト〉，〈ボリュームタイプ〉，〈メモ〉，〈パスワード〉，〈ボリューム名〉によって指定された現在の保護空間内の仮想ボリュームの一覧表を，書式1）はCRT画面に，書式2）はプリンタに出力します.

〈モードバイト〉には出力したい仮想ボリュームの属性などを指定します．指定する値と出力されるボリュームは次のとおりです.

省略　：　保護空間内の全ボリューム
L　：　LOCALボリューム
W　：　WORKボリューム
P　：　PUBLICボリューム
S　：　SYSTEMボリューム
A　：　アクティブボリューム（ドライブに接続されているボリューム）

N　：　LOCALボリュームおよびWORKボリューム
G　：　SYSTEMボリュームおよびPUBLICボリューム

〈ボリュームタイプ〉には出力するボリュームタイプを指定します．指定する値と出力されるボリュームのタイプは次のとおりです．

省略　：　すべてのボリュームタイプ
1　：　160KBタイプフロッピィディスク
2　：　320KBタイプフロッピィディスク
3　：　640KBタイプフロッピィディスク
S　：　1MBタイプフロッピィディスク
E　：　5インチ固定ディスク（5MB）
F　：　5インチ固定ディスク（10MB）
G　：　5インチ固定ディスク（20MB）

〈メモ〉，〈パスワード〉，〈ボリューム名〉を指定して，一致するものだけを出力させることもできます．

ボリューム名にはワイルドカードキャラクタ"?"が使えます．ワイルドカードキャラクタ"?"は，文字列の中の？の位置に任意の1文字またはヌル文字（空の文字列）があてはまることを意味します．たとえばボリューム名に"ABC???"と指定すれば，"ABC123"，"ABCDEF"のように6桁のうちはじめ3桁が"ABC"のボリューム名をもつ仮想ボリュームの一覧表を表示することができます．

CMD　VOLSによって見ることのできる仮想ボリュームは，SYSTEM，PUBLICボリュームのほか，現在のユーザー名に対応した保護空間内で作成されたLOCAL，WORKボリュームです．他の保護空間内で作成されたLOCAL，WORKボリュームは見ることができません．

CMD　VOLS／LVOLSを実行すると，CRT画面もしくはプリンタには，ボリュームの通し番号，ボリューム名，属性，ボリュームタイプ，メモ，ステーション番号，使用サイズ，作成日，現在の状態が出力されます．

このうち，ボリュームタイプについては“Type”の欄に次のような記号で示されます．

160KB　：　160KBタイプフロッピィディスク
320KB　：　320KBタイプフロッピィディスク
640KB　：　640KBタイプフロッピィディスク
1MB　：　1MBタイプフロッピィディスク
5HD　：　5インチ固定ディスク（5MB）
10HD　：　5インチ固定ディスク（10MB）
20HD　：　5インチ固定ディスク（20MB）

注意：表示されるボリュームの通し番号は便宜的なもので，ボリューム名の代わりとして用いることはできません．
複数個のパラメータを指定した場合は，すべての条件を満たすものだけが表示されます．

# 〔1〕COLOR　C

機　能　ディスプレイ画面の各部の色およびグラフィック画面のパレットモードを指定します．

書　式　COLOR　〔<ファンクションコード>〕〔，<バックグラウンドカラー>〕〔，<ボーダーカラー>〕〔，<フォアグラウンドカラー>〕〔，<パレットモード>〕

文　例　COLOR　7，0，0，7

〔1〕COLORは，テキスト画面の文字のカラーを変えたり，グラフィック画面のフォアグラウンド，バックグラウンド，ボーダーの各カラーを設定したり，グラフィック画面のパレットモードを指定したりするのに使用します．

<ファンクションコード>は，テキスト画面の文字にいろいろな機能を与えます．このファンクションコードは，テキスト画面がカラーモードになっているか，白黒モードになっているかによって，働きが異なります．なお，カラーモード／白黒モードの切り替えはCONSOLEで行います．<ファンクションコード>で指定する値は，〝パレット番号〟ではありませんので，〔2〕COLORによってカラーパレットの変更を行っても色は変化しません．

白黒モードの場合(CONSOLE　，，，0)

0　：　ノーマル(通常の表示)
1　：　シークレット(文字は表示されない)
2　：　ブリンク(点滅する)
3　：　シークレット(1と同じ)
4　：　リバース(反転する)
5　：　リバースシークレット(反転して文字は表示されない)
6　：　リバースブリンク(反転して点滅する)
7　：　リバースシークレット(5と同じ)

カラーモードの場合(CONSOLE　，，，1)

0　：　黒　　1　：　青　　2　：　赤　　3　：　紫
4　：　緑　　5　：　水色　　6　：　黄　　7　：　白

<バックグラウンドカラー>は，グラフィック画面の地の色を表します．このパラメータは，〝パレット番号〟(<パレットモード>の項および〔2〕COLOR参照)によって指定します．この

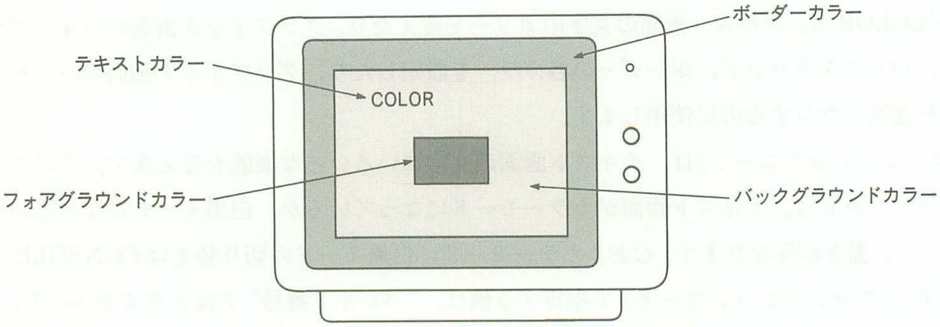
パラメータの設定後CLSによって画面をクリアすると，設定色によって画面がぬり変えられます．また，以後PRESETを色指定なしで実行すると，この色が採用されます．

なお，白黒モード(高分解能白黒モードを含む)では，〈バックグラウンドカラー〉として指定されたパレット番号は，0のときは黒，0以外のときは白とみなされます．

〈ボーダーカラー〉は，画面の中でBASICによって使うことのできる領域外の，枠の色を表すもので，次のような値で指定します．

| | | | |
|---|---|---|---|
| 0 : 黒 | 1 : 青 | 2 : 赤 | 3 : 紫 |
| 4 : 緑 | 5 : 水色 | 6 : 黄 | 7 : 白 |

〈ボーダーカラー〉が指定されると，カラーモードでも白黒モードでも，画面の外枠に色(白黒のディスプレイでは濃淡)がついて表示されます．なお，このパラメータは，専用高解像度ディスプレイの使用時には意味がありません．




〈フォアグラウンドカラー〉は，グラフィック画面に点や線を表示したりするときに使われる色を表します．このパラメータは，〝パレット番号〞によって指定します．種々のグラフィック命令(PSET, LINE, CIRCLEなど)でとくに色指定をしなかった場合，この色が採用されます．

なお，白黒モード(高分解能白黒モードを含む)では，〈フォアグラウンドカラー〉として指定されたパレット番号は，0のときは黒，0以外のときは白とみなされます．

〈パレットモード〉は，グラフィック画面に対する色指定のモードを指定します．〈パレットモード〉に指定できる値とその意味は次のとおりです．

0 : 8色中•8色モード．8個のパレットにシステムが決めた8色を対応づけるモードです．

1 : 4096色中•8色モード．8個のパレットに4096色中の任意の8色を対応づけるモードです．

2 : 4096色中•16色モード．16個のパレットに4096色中の任意の16色を対応づけるモードです．

$N_{88}$-日本語BASIC(86)(MS-DOS版)の起動直後のパレットモードは8色中•8色モードで

す．以降，〈パレットモード〉が指定された場合に限りパレットモードが切り替わります．各モードにおけるパレットとカラーコードの対応づけは，〔2〕COLOR によって設定されますので，参照してください．

〈パレットモード〉が指定されてパレットモードが切り替わると，そのたびにパレットとカラーコードの関係は次のように初期化されます．

**8 色中･8 色モード**

| 〈パレット番号〉 | 〈カラーコード〉 | |
|---|---|---|
| 0 | 0 | （黒） |
| 1 | 1 | （明るい青） |
| 2 | 2 | （明るい赤） |
| 3 | 3 | （明るい紫） |
| 4 | 4 | （明るい緑） |
| 5 | 5 | （明るい水色） |
| 6 | 6 | （明るい黄） |
| 7 | 7 | （白） |

**4096 色中･8 色モード**

| 〈パレット番号〉 | 〈カラーコード〉 | |
|---|---|---|
| 0 | &H000 | （黒） |
| 1 | &H00F | （明るい青） |
| 2 | &H0F0 | （明るい赤） |
| 3 | &H0FF | （明るい紫） |
| 4 | &HF00 | （明るい緑） |
| 5 | &HF0F | （明るい水色） |
| 6 | &HFF0 | （明るい黄） |
| 7 | &HFFF | （白） |

**4096 色中･16 色モード**

| 〈パレット番号〉 | 〈カラーコード〉 | | 〈パレット番号〉 | 〈カラーコード〉 | |
|---|---|---|---|---|---|
| 0 | &H000 | （黒） | 8 | &H777 | （灰色） |
| 1 | &H00F | （明るい青） | 9 | &H00A | （少し暗い青） |
| 2 | &H0F0 | （明るい赤） | 10 | &H0A0 | （少し暗い赤） |
| 3 | &H0FF | （明るい紫） | 11 | &H0AA | （少し暗い紫） |
| 4 | &HF00 | （明るい緑） | 12 | &HA00 | （少し暗い緑） |
| 5 | &HF0F | （明るい水色） | 13 | &HA0A | （少し暗い水色） |
| 6 | &HFF0 | （明るい黄） | 14 | &HAA0 | （少し暗い黄） |
| 7 | &HFFF | （白） | 15 | &HAAA | （少し暗い白） |

**注意：** 4096 色中･8 色モードおよび 4096 色中･16 色モードはアナログ RGB 対応ディスプレイが接続されている場合のみ有効です．アナログ RGB 対応ディスプレイが接続されていない場合でも，エラーにはなりませんが，指定どおりの色は出ません．
さらに，4096色中・16色モードの指定は16色グラフィックスボードが実装されている場合のみ有効です．16色グラフィックスボードが実装されていない場合，このモードを指定するとエラーになります．

**参照：** 〔2〕COLOR，COLOR＠，CONSOLE，サンプルプログラム15，16

# 〔2〕COLOR　C

**機　能**　カラーパレットの色を変更します.

**書　式**　COLOR〔=(〈パレット番号〉, 〈カラーコード〉)〕

**文　例**　COLOR=(2, 4)

COLOR=(PAL, COL)

グラフィック画面への色の指定は，すべてカラーパレットによって行いますが，〔2〕COLORは，このカラーパレットの色を決める命令です.

〈パレット番号〉には0から7(あるいは0から15)の8個(あるいは16個)のカラーパレットの固有の番号を指定します.

〈カラーコード〉には，〈パレット番号〉にどの色を対応づけるかを指定します.

パレットの個数およびカラーコードの値は，パレットモード(〔1〕COLOR参照)によって異なります. 各モードにおける〈パレット番号〉と〈カラーコード〉の関係は次のとおりです.

### 8色中・8色モード

0～7の〈パレット番号〉に0～7の8個の〈カラーコード〉を任意に指定します.

### 4096色中・8色モード

0～7の〈パレット番号〉に&H000～&HFFFの4096個の〈カラーコード〉から任意に指定します.

### 4096色中・16色モード

0から15の〈パレット番号〉に&H000～&HFFFの4096個の〈カラーコード〉から任意に指定します.

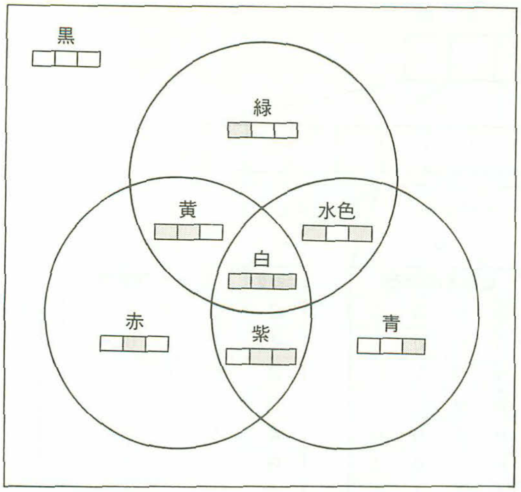
どのモードにおいても，パレット番号とカラーコードは任意に対応させることができます. また，同じカラーコードを複数のパレットに対応づけてもかまいません.

等号(=)，〈パレット番号〉および〈カラーコード〉を省略して，"COLOR"とすると，パレットとカラーコードの関係を初期化することができます. パレットとカラーコードの初期状態に関しては，〔1〕COLORを参照してください.

ここで，4096色中・8モードおよび4096色中・16色モードにおけるカラーコードの値と色の関係について説明します.

(1)　色のしくみ

グラフィック画面に表示できる色はすべて基本色である緑(G)，赤(R)，青(B)の組み合わせで表現されます.




G R B：左から緑(G)，赤(R)，青(B)に対応しています．■はONの状態を，□はOFFの状態を示しています．

たとえば，水色の場合 G R B は，BとGがONの状態です．すなわち，水色は青と緑を混ぜて表現される色であることを示しています．

(2) 4096種類の色

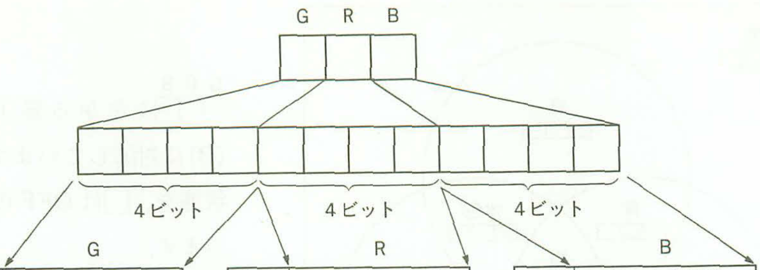
黒，青，赤，紫，緑，黄，水色，白の色のしくみは(1)で示したとおりですが，それではオレンジ色や黄緑などの中間色はどのように表現するのかを説明します．たとえば，オレンジ色というのは，黄に赤を混ぜた色です．黄は赤と緑を混ぜた色ですが，赤と緑を混ぜあわせる際に緑よりも赤を多く混ぜ合わせるとオレンジ色になります（赤と緑を同じ程度に混ぜると黄になります）．

色を混ぜ合わせる割り合いは，「輝度」によって調整することができます．「輝度」はその名のとおり明るさの度合いを示したものです．G，R，Bそれぞれ16段階の輝度を表現することができます．G，R，Bはそれぞれ4ビットの情報を持ち，このビットの状態で輝度を表しています（16段階の輝度を表すのには16進表現が便利です）．

輝度が0というのは，その色が表示されないことを意味します．オレンジ色を表現する場合は，輝度が&H9の緑(暗い緑)と輝度が&HFの赤(最も明るい赤)を混ぜ合わせることによって表現します（カラーコードには&H9F0と指定します）．このように輝度を変えることで，様々な色合いが表現できます．また，基本の7色についてもその輝度を変えれば，暗い赤や，暗い黄などの表現ができます．

例）　明るい黄──→輝度が&HFの緑と&HFの赤の混ぜ合わせ
（カラーコードには&HFF0と指定します）

暗い黄　──→輝度が&H7の緑と&H7の赤の混ぜ合わせ
（カラーコードには&H770と指定します）




G

| 輝度 | ビットの状態 |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

R

| 輝度 | ビットの状態 |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

B

| 輝度 | ビットの状態 |
|---|---|
| 0 | 0 0 0 0 |
| 1 | 0 0 0 1 |
| 2 | 0 0 1 0 |
| 3 | 0 0 1 1 |
| 4 | 0 1 0 0 |
| 5 | 0 1 0 1 |
| 6 | 0 1 1 0 |
| 7 | 0 1 1 1 |
| 8 | 1 0 0 0 |
| 9 | 1 0 0 1 |
| A | 1 0 1 0 |
| B | 1 0 1 1 |
| C | 1 1 0 0 |
| D | 1 1 0 1 |
| E | 1 1 1 0 |
| F | 1 1 1 1 |

暗い ↓ 明るい

緑(G)と赤(R)と青(B)はそれぞれ16段階の輝度が表現できます．そして，これらの色の混ぜ合わせを自由に行うと16×16×16＝4096種類の色が表現できます．

参照：〔1〕COLOR，サンプルプログラム15，16

# COLOR@ 〔C〕

| | |
|---|---|
| 機能 | テキスト画面に書かれた文字などに色や機能を設定します． |
| 書式 | COLOR@　(X1，Y1)－(X2，Y2)〔，<ファンクションコード>〕 |
| 文例 | COLOR@　(0，0)－(79，4)，2 |

COLOR@は，テキスト画面のキャラクタ座標の2点(X1，Y1)，(X2，Y2)を対角とする四角形の領域に書かれている，文字やグラフィックキャラクタに色をつけたり，ブリンクなどの機能を設定したりします．

(X1，Y1)，(X2，Y2)は，必ずキャラクタ座標でなくてはなりません．

<ファンクションコード>は，CONSOLEによって設定されているモードによって持つ意味が異なります．この機能および指定の仕方は〔1〕COLORの<ファンクションコード>の場合とまったく同様ですのでそちらを参照してください．省略された場合は，7を値として採用しま

す．

この命令は，テキスト画面に書かれている文字などに対して有効です．したがって何も書かれていない場合は，何の作用もありません．またこの命令を実行した領域の上に新しく文字などを書いた場合，書かれた文字はこの命令の影響を受けません．

参照：〔1〕COLOR，CONSOLE，サンプルプログラム33

# COMMON

Ⓒ

機能　CHAINが実行された際，メモリ上のプログラムから，実行の移されたプログラムに変数を引き渡します．

書式　COMMON　<変数名>〔，<変数名>…〕

文例　COMMON　A，B，XY()，NA$

COMMONは，CHAINによってメモリ上のプログラムからディスク上のプログラムに実行を移す際，メモリ上のプログラムの変数を実行の移されたプログラムに引き渡します．なお，CHAINでMERGEオプションが指定されたときには，メモリ上のプログラムとディスク上のプログラムとが連結された後のプログラムに変数を引き渡します．

したがって，COMMONは必ずCHAINとあわせて使われます．また，プログラム内において，COMMONはCHAINの前にある必要があります．

<変数名>は1行の許す範囲(255バイト以内)で何個でも並べることができますが，1つのプログラム中のCOMMONで同じ変数名を指定してはなりません．また配列変数名は()を後ろにつけ加えて表現します．

すべての変数を引き渡したいのであれば，インタプリタの場合CHAINのALLオプションを使う方が便利です(コンパイラにはこの機能はありません)．

コンパイラの場合，結合されるプログラム側にもCOMMONが必要です．また，呼び出す側のCOMMONの変数名指定順序および変数の個数と呼び出される側(連結される側)のそれは同じでなければなりません．

参照：CHAIN，サンプルプログラム1，2

# COM ON／OFF／STOP　C

機能　RS-232C 回線からの割り込みの許可，禁止，停止を制御します．

書式　1) COM〔(〈回線番号〉)〕　ON

2) COM〔(〈回線番号〉)〕　OFF

3) COM〔(〈回線番号〉)〕　STOP

文例　COM　ON

COM(2)　ON

RS-232C 回線に外部からの通信が入ったことによる割り込みの許可，禁止，停止を宣言します．

〈回線番号〉に指定できる値と意味は次のとおりです．(〈回線番号〉)を省略(カッコも含む)した場合は，すべての回線が処理対象となります．

1　：　RS-232C 第1回線
2　：　RS-232C 第2回線
3　：　RS-232C 第3回線

ただし，2，3は専用のインターフェイスボードが必要です．

書式1）割り込みを許可します．以後，〈回線番号〉で指定した RS-232C 回線に受信が入るごとに割り込みが発生し，ON　COM　GOSUB によって定義された処理ルーチンに分岐します．

書式2）割り込みを禁止します．以後，通信があっても処理ルーチンへの分岐は起こりません．

書式3）割り込みを停止します．以後，通信があってもそのことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後，COM　ON によって割り込みが許可されると，前に通信があったことによって，処理ルーチンに分岐します．

注意：プログラムの終了時には COM　OFF を実行しておいてください．COM　ON は，RS-232C 回線を OPEN した後実行されねば有効となりません．

なお，割り込み処理ルーチンに制御が移ると自動的に割り込み停止状態となりますので，割り込み処理ルーチンの先頭では COM　OFF あるいは COM　STOP を実行する必要はありません．

参照：ON　COM　GOSUB

# CONSOLE　C

機能　テキスト画面モードの設定を行います.

書式　CONSOLE 〔<スクロール開始行>〕〔, <スクロール行数>〕〔, <ファンクションキー表示スイッチ>〕〔, <カラー／白黒スイッチ>〕〔, <キャラクタモード>〕

文例　CONSOLE 0, 24, 0, 1
　　　CONSOLE ,, 1, 0

<スクロール開始行> で指定した行以降の, <スクロール行数> で指定した行数分を, 画面上でスクロールする領域(スクロールウィンドウ)として設定します. 画面のクリア(CLS あるいは PRINT　CHR$(12))は, このスクロールウィンドウに対して実行されます.

<ファンクションキー表示スイッチ>に1を指定すると, 画面最下行にファンクションキーに定義されている文字列を表示します. 0を指定すると, 表示をとりやめます. インタプリタの場合, 起動直後は表示される状態になっています.

コンパイラでコンパイルされたプログラムの場合, 起動直後は表示されないモードになっています.

<カラー／白黒スイッチ> に1を指定すると, テキスト画面をカラーモードにし, 0を指定すると白黒モードにします. 起動直後は白黒モードになっています.

<キャラクタモード>に1を指定すると, グラフィックキャラクタモードになります. グラフィックキャラクタモードにおいては日本語を画面に表示したり, プリンタに印字したりすることはできませんが, グラフィックキャラクタ(キャラクタコード&H81～&H9F, &HE0～&HFC)の表示／印字が可能になります. 0を指定すると, 日本語キャラクタモードになります. 日本語キャラクタモードにおいてはグラフィックキャラクタの表示／印字は行えません. 起動直後は日本語キャラクタモードになっています.

参照: 〔1〕COLOR, COLOR@, サンプルプログラム33

# CONT

機能　STOP キーあるいは CTRL ＋ C の入力, または STOP によって停止したプログラムの実行を再開します.

書式　CONT

文例　CONT

CONT は通常デバッグのために用いられます．STOP キー（または CTRL ＋ C ）の入力や，プログラム中のSTOPの実行によってプログラムの実行を停止すると，ダイレクトモードで停止時における変数の内容などを調べることができます．この後，CONT を実行することにより，プログラムの実行を再開することができます．

コンパイラにおいてはこの機能は使用できません．

**注意**：実行停止中にプログラム内容の変更を行った場合には，CONT による継続実行はできません．また，実行停止のタイミングによってはプログラムの継続ができない場合もあります．

**参照**：RUN，STOP

# COPY C

| 機能 | 画面情報のハードコピーを行います． |
|---|---|
| 書式 | COPY　〔<機能>〕 |
| 文例 | COPY 2<br>COPY |

2 種類の画面ハードコピー機能が用意されています．

**(1) メモリスイッチ SW6 の $2^4$ ビットが OFF の場合．**

<機能>に指定できる値は1〜5で，それぞれ次のように働きます．<機能>が省略された場合，3を値として採用します．

なお，4，5の機能はカラー出力の場合のみ有効です．

1 ： テキスト画面のみを出力．
2 ： グラフィック画面のみを出力（カラー出力の場合，白黒が反転される）．
3 ： テキスト画面とグラフィック画面を合成して出力（カラー出力の場合，白黒が反転される）．
4 ： グラフィック画面のみを出力（白黒は反転されない）（カラー出力用）．
5 ： テキスト画面とグラフィック画面を合成して出力（白黒は反転されない）（カラー出力用）．

PC-PR601系ページプリンタを使用する場合，上記の1〜3の機能に対して，印字方向や印字倍率を指定することができます．詳しくは $N_{88}$-日本語 BASIC(86)(MS-DOS版)ユーザーズマニュアルを参照してください．

注意：PC-PR201系プリンタまたはPC-PR601系ページプリンタを使用する場合，メモリスイッチSW5の$2^0$ビットをONにしてください.

また，カラー出力を行う場合，次の準備が必要です.

- PC-PR201V系カラープリンタを接続する
- メモリスイッチSW6の$2^3$ビットをONにする

なお，仮想プリンタに対しては，テキスト画面しか出力しません.

(2) メモリスイッチSW6の$2^4$ビットがONの場合.

<機能>に指定できる値は1～5で，それぞれ次のように働きます.<機能>が省略された場合，3を値として採用します.

1 ： テキスト画面のみを出力.

2 ： グラフィック画面のみを出力.

3 ： テキスト画面とグラフィック画面を重ね合わせて出力.このときテキスト画面の情報はプリンタの印字体で出力される.

4 ： グラフィック画面のみを出力. 640×200モードの場合に有効.

5 ： テキスト画面とグラフィック画面を重ね合わせ，さらに縦方向に縮小して出力.

注意：PC-PR201系プリンタまたはPC-PR601系ページプリンタを使用する場合，メモリスイッチSW5の$2^0$ビットをONにしてください.

## COS 関数 C

機能　余弦(コサイン)を得ます.

書式　COS(<数式>)

文例　X=RADIUS * COS(ANGLE)

PRINT　COS(3.14159/4)

<数式>の値に対する余弦値を得ます.<数式>の単位はラジアンで指定します.

<数式>に倍精度実数が含まれる場合，得られる値は倍精度となりますが，他の場合には単精度となります.

参照：ATN，SIN，TAN，サンプルプログラム6，30

# CSNG 関数 [C]

| | |
|---|---|
| 機能 | 整数値，倍精度実数値を，単精度実数値に変換します． |
| 書式 | CSNG(<数式>) |
| 文例 | A!=CSNG(B#/4)<br>PRINT　CSNG(128%) |

<数式>の値を有効数字6桁の単精度実数値に変換した値を得ます．結果の値が－1.70141E＋38～1.70141E＋38の範囲にない場合は，"Overflow" エラーになります．

参照：CINT，CDBL

# CSRLIN 関数 [C]

| | |
|---|---|
| 機能 | 現在のカーソルの行位置を得ます． |
| 書式 | CSRLIN |
| 文例 | Y=CSRLIN |

現在のカーソルの行(垂直)位置をキャラクタ座標で得ます．値の範囲は25行モードでは0～24，20行モードでは0～19となります．得られる値は，画面の最上行が0，最下行が24(または19)を意味します．

参照：POS，サンプルプログラム10

# CVI／CVS／CVD 関数 [C]

| | |
|---|---|
| 機能 | 文字列を数値データに変換します． |
| 書式 | 1) CVI(<2文字の文字列>)<br>2) CVS(<4文字の文字列>)<br>3) CVD(<8文字の文字列>) |
| 文例 | A%=CVI(A$)<br>B=CVS("A3Bd")<br>C#=CVD(NUM$) |

これらの関数は，MKI$／MKS$／MKD$の各関数を用いてランダムディスクファイルに書き込まれた文字型化数値データを，数値データにもどす際に使用します．

CVI　：　2文字(2バイト)の文字列を整数値に変換(MKI$の逆).

CVS　：　4文字(4バイト)の文字列を単精度実数値に変換(MKS$の逆).

CVD　：　8文字(8バイト)の文字列を倍精度実数値に変換(MKD$の逆).

実際にランダムディスクファイルから数値データを読み込む際には，まずGETでフィールド変数に文字型化数値データを読み込み,次にこれらの関数を用いて数値データに変換します.

参照：MKI$／MKS$／MKD$，サンプルプログラム27

# DATA　C

| | |
|---|---|
| 機　能 | READで読み込まれる数値定数，文字定数を定義します. |
| 書　式 | DATA　<定数>〔，<定数>…〕 |
| 文　例 | DATA　1，CBA，1465 |

<定数>には文字型か数値型のデータを指定します．READで読み込む場合，DATAで指定した文字定数は文字変数に読み込まなくてはなりませんが，数値定数は文字変数，数値変数のいずれに読み込むこともできます．但し，定数式は使えません．また数値には10進形式以外は指定できません.

DATA行中には,1行(255バイト)に入るだけのデータをセットすることができます.データはコンマ(,)によって区切りますが，文字定数で，その中にコンマやピリオド(.)，意味のある空白を置きたい場合，および日本語文字を含む文字列である場合は，その文字定数の前後をダブルクォーテーション(")でくくる必要があります.

1つのプログラムには任意の数のDATA行を置くことができます．DATA行はプログラム中のどこに置いてもかまいません.READは行番号の小さい方から順番に,DATA行中のデータを読み込んでいきます.

参照：READ，RESTORE，サンプルプログラム3，14

# DATE$　関数　C

| | |
|---|---|
| 機　能 | 日付を得ます. |
| 書　式 | 1) DATE$<br>2) DATE$＝"yy／mm／dd" |
| 文　例 | A$＝DATE$<br>DATE$＝"87／12／12" |

DATE$には常に現在の日付が"yy／mm／dd"(年／月／日)の形で入れられており，いつでもその内容を見ることができます．

なお，書式 2)を用いることにより，日付を変更することもできます．yy には 00～99，mm には 01～12，dd には 01～31 の範囲の整数値を指定します．yy，mm，dd の間はスラッシュ(／)で区切ってください．

注意：日付は，バッテリーバックアップによって自動的に更新され，正しく維持されるようになっています．不用意に日付を変えないようにしてください．ただし，PC-9801UV21／VM21, PC-9801VX(各機種), PC-9801UX(各機種), PC-98XL(ノーマルモード)以外の機種を御使用の場合，年は自動的に更新されませんので，DATE$で更新する必要があります．また，2月は 28 日単位で更新されますので，うるう年については 2 月 29 日以降 DATE$による日付の再設定が必要です．

参照：TIME$

# DEF FN　Ⓒ

| | |
|---|---|
| 機能 | 利用者定義関数を指定します． |
| 書式 | DEF　FN〈名前〉〔(〈パラメータリスト〉)〕＝〈関数の定義式〉 |
| 文例 | DEF　FNA(X，Y)＝X＊2＋Y＊3 |

DEF　FN は，利用者が使うことのできる関数を定義する命令です．定義する関数は，数値関数，文字関数，その混在のいずれでもかまいません．

DEF　FN はプログラム中でしか実行することはできません．

〈名前〉は変数名として正しい形をしたものでなければなりません．

〈パラメータリスト〉には，〈関数の定義式〉の中で使われている変数とおなじ名前の変数を用います．この変数名は〈関数の定義式〉を評価する際にのみ有効なものであり，プログラム中に同一名の変数があってもかまいません．

〈関数の定義式〉は，その関数の演算内容を記述する式で，1 行の範囲(255 バイト以内)に限られます．

定義した関数は「FN 名前(変数)」という形で呼び出して使います．定義したときの変数は仮のものであるため，呼び出すときにはその時々で必要な変数に変えて指定することができます．ただし，変数の型は同一でなければなりません．

注意：DEF　FN の〈関数の定義式〉中で使われている変数が，〈パラメータリスト〉内にない場合は，その変数がその時点で持っている値が使われます．

DEF　FN は，これによって定義される関数がプログラム中で呼ばれる前に実行されて

いなければなりません．

DEF　FNは，DEFとFNの間に必ずスペースを入れなければなりません．スペースを省略してDEFFNとすると変数として扱われてしまいます．

参照：サンプルプログラム39

# DEFINT／DEFSNG／DEFDBL／DEFSTR　Ⓒ

**機　能**　変数の型宣言を行います．

**書　式**

1) DEFINT　〈文字の範囲〉〔,〈文字の範囲〉…〕

2) DEFSNG　〈文字の範囲〉〔,〈文字の範囲〉…〕

3) DEFDBL　〈文字の範囲〉〔,〈文字の範囲〉…〕

4) DEFSTR　〈文字の範囲〉〔,〈文字の範囲〉…〕

**文　例**

```
DEFINT  A, I-K
DEFSNG  B
DEFDBL  X-Z
DEFSTR  L-N, Q
```

〈文字の範囲〉で指定された文字で始まる変数を，DEFINTでは整数型に，DEFSNGでは単精度実数型に，DEFDBLでは倍精度実数型に，DEFSTRでは文字型にそれぞれ定義します．

〈文字の範囲〉で指定できる文字は英字1文字で，複数個指定する場合はマイナス記号でつないでその範囲を示します．

注意：この命令によって行われる型宣言より，型宣言文字による指定(%，!，#，$)の方が優先されます．また，型宣言が行われていない文字で始まり，型宣言文字による指定もされていない変数は，すべて単精度変数とみなされます．

コンパイラにおいては，これらの型宣言文は非実行文として扱われます．すなわち，型宣言文が実行された時点で動的に変数の型が決まるのではなく，コンパイル時に変数の型が決まります．したがって，コンパイルされたプログラム中で変数の型を再変更することはできません．

参照：サンプルプログラム39

# DEF SEG　Ⓒ

| | |
|---|---|
| 機　能 | セグメントベースを宣言します. |
| 書　式 | DEF　SEG=〈セグメントベース〉 |
| 文　例 | DEF　SEG=SEGPTR(2) |

BSAVE, BLOAD, PEEK, POKEなどの命令でメモリのアドレスを指定する場合には，まず，アクセスしようとするメモリ空間を含むセグメントを，DEF　SEGであらかじめ宣言する必要があります．そして，実際のメモリのアドレスの指定の際には，該当するセグメント内での相対アドレスで指定します．

〈セグメントベース〉には，物理アドレスを16で割った値を指定します．

プログラム実行時における種々の制御領域のセグメントベースは，SEGPTRにより得ることができます．

また，各変数や各配列が記憶されているセグメントベースは，VARPTRにより得ることができます．なお，配列については配列ごとに別セグメントとなっていますので，複数の配列の要素あるいは配列要素と変数にアクセスする場合には，そのつどセグメントベースを切り換えてアクセスすることが必要となります．

注意：DEF　SEGは，DEFとSEGの間に必ずスペースを入れなければなりません．スペースを省略してDEFSEGとすると変数として扱われてしまいます．

参照：BLOAD, BSAVE, CLEAR, DEF　USR, PEEK, POKE, SEGPTR, VARPTR

# DEF USR　Ⓒ

| | |
|---|---|
| 機　能 | USRで呼び出す機械語関数の番号と実行開始アドレスを定義します. |
| 書　式 | DEF　USR[〈番号〉]=〈開始アドレス〉 |
| 文　例 | DEF　USR3=&HF000 |

〈番号〉は，0～9までの値で，複数の機械語関数を用いる場合の識別を行います．最大10個までの機械語関数を準備，利用することができます．〈番号〉が省略された場合には0と解釈されますので，USR0とUSRは同じ意味となります．

〈開始アドレス〉は，〈番号〉によって指定される機械語関数の実行開始番地を，直前に実行されたDEF　SEGで指定されたセグメントベースからの相対アドレスで指定します．

なお，USRで呼び出したい機械語関数は，あらかじめメモリ上に用意しておかなくてはなりません．このためには，POKEやBLOADを用いることができます．

注意：DEF　USR は，DEF と USR の間に必ずスペースを入れなければなりません．スペースを省略して DEFUSR とすると変数として扱われてしまいます．

参照：BLOAD，BSAVE，CLEAR，DEF SEG，POKE，USR

# DELETE

機　能　プログラムの部分削除を行います．

書　式　DELETE ｛〈始点行番号〉〔－〈終点行番号〉〕
　　　　　　　　 〔〈始点行番号〉〕－〈終点行番号〉｝

文　例　DELETE　1050

DELETE　＊START-＊SUB

DELETE　.－500

〈始点行番号〉から〈終点行番号〉までのプログラムを削除します．〈始点行番号〉だけを指定した場合はその行だけを削除し，マイナス記号以下〈終点行番号〉を指定した場合には，プログラムの先頭から指定行までを削除します．

なお，〈始点行番号〉および〈終点行番号〉の両方とも省略することはできません．

〈行番号〉にピリオド(.)を指定すると BASIC インタプリタ内のポインタが示している現在行を指定したことになります．実際には，LIST コマンドで最後に表示された行や，編集機能で最後に修正された行が現在行となります．

コンパイラにおいてはこの機能は使用できません．

参照：LIST

# DIM　Ⓒ

機　能　配列変数の要素の大きさを指定し，メモリ領域に割り当てます．

書　式　DIM　〈変数名〉(〈添字の最大値〉〔，〈添字の最大値〉…〕)〔，〈変数名〉(〈添字の最大値〉〔，〈添字の最大値〉…〕)…〕

文　例　DIM　A(12，2)，B$(3)

DIM は，配列変数の添字の最大値を設定し，同時にメモリ上にその配列の領域を割り当てます．

〈変数名〉には，変数名として正しい形をしたものを指定します．

〈添字の最大値〉には，配列の要素を示す添字の，最大値を指定します．添字の最小値は，OPTION　BASEによって，0か1に指定することができます．実際に割り当てられる配列の要素の数は，OPTION　BASEが1のときは〈添字の最大値〉に等しく，OPTION　BASEが0のときは〈添字の最大値〉+1となります．

なお，OPTION　BASEを指定しないでDIMを使用した場合，添字の最小値は0となります．

〈添字の最大値〉を複数個指定すると，個数分の次元をもつ配列変数の指定となります．1行中(255バイト)に表記できる範囲であれば，次元数はいくつとってもかまいません．

ただし，配列の要素数および次元数は，メモリ容量の制限を受けます．配列変数のとるメモリ領域が大きすぎると，〝Out　of　memory〞エラーとなります．

数値配列の場合，1個の配列に対し指定できる要素の数は次のとおりです．

| | | |
|---|---|---|
| 整数型の場合 | ： | 32767個 |
| 単精度実数型の場合 | ： | 16383個 |
| 倍精度実数型の場合 | ： | 8191個 |

DIMが実行された時点で，その配列のすべての要素の値は0(文字型の場合はヌルストリング)に設定されます．

配列変数を消去するには，ERASEを用います．

注意：設定された最大値より大きな値の添字が用いられた場合は，〝Subscript out of range〞エラーが起こります．
DIMで宣言しなくとも配列変数を用いることができますが，その場合，添字の最大値は10まで(OPTION　BASEが0でも1でもかまわない)使用できます．

参照：ERASE，OPTION　BASE，サンプルプログラム5

# DRAW　C

機能　グラフィック描画サブコマンド列に従って，ワールド座標上で図形を描きます．

書式　DRAW　〈文字列式〉

文例　DRAW　"D50R50U50L50"
　　　DRAW　"C7T150，50Z=TILE$；"

DRAWでは，複数のグラフィック描画サブコマンドを使って図形を描きます．〈文字列式〉内に指定可能なグラフィック描画サブコマンドは，一般的には次のような形式です．

〔〈修飾子〉〕〈識別子〉〔〈パラメータ1〉〕〔，〈パラメータ2〉〕…

### (1) 修飾子

〈修飾子〉としては，BおよびNを指定することができます．〈修飾子〉は〈識別子〉の補助的な役割を果たすもので，次のような意味をもちます．

B　〈識別子〉で示されるサブコマンドの実行は実際には行わず，LP(最終参照点)のみをサブコマンド実行後の値にします．

N　〈識別子〉で示されるサブコマンド実行後，LPを(0, 0)とします．

なお，BおよびNを同時に指定した場合は，〈識別子〉で示されるサブコマンドの実行は行われず，LPが(0, 0)となります．

### (2) 識別子とパラメータ

〈識別子〉は，いろいろな描画機能を表すもので，英字1文字のサブコマンドで指定します．

〈パラメータ1〉，〈パラメータ2〉は，各サブコマンドに対応した数値定数または変数で，以下の形式で指定します．

**〈数値定数〉〔；〕**

または，

**＝〈変数〉；**

数値定数として指数形式を用いることはできません．また，16進表記で数値定数を表す場合には，その後に必ず，セミコロン(；)をつけなければなりません．

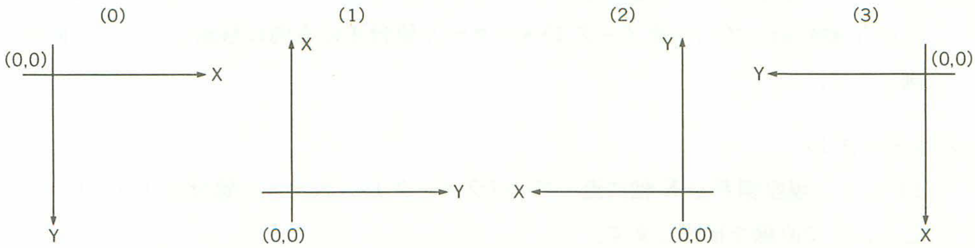
変数を指定する場合は，その直前には等号(＝)を，直後にはセミコロン(；)を，それぞれ必ずつけるようにしてください．

> コンパイラにおいては＝〈変数〉；指定は使用できません．

〈識別子〉に指定するサブコマンドとその〈パラメータ〉について次に説明します．

A〈パラメータ1〉

グラフィック描画サブコマンドの現座標系を，〈パラメータ1〉に0〜3までの整数値を指定することにより，以下の状態に規定します．



Aサブコマンドによって規定された座標系はDRAWのグラフィック描画サブコマンドに対してのみ有効なもので，ワールド座標系が変化するわけではありません．

初期状態では，〈パラメータ1〉の値は0となっています．

C〈パラメータ1〉

グラフィック描画サブコマンドによって描画される図形のパレット番号を規定します．〈パラメータ1〉にはパレット番号を指定します．Cサブコマンドによって規定されたパレット番号は，グラフィック描画サブコマンドに対して有効なものであり，フォアグラウンドカラーを変更するものではありません．

初期状態では，〈パラメータ1〉の値はフォアグラウンドカラーとなっています．

D〈パラメータ1〉

LPから，現座標系のY軸に沿って〈パラメータ1〉×スケール値分だけ正の方向へ移動した点まで直線を描画します．スケール値については，Sサブコマンドを参照してください．

E〈パラメータ1〉

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分だけ正の方向に，さらにY軸に沿って〈パラメータ1〉×スケール値分負の方向に移動した点まで直線を描画します．

F〈パラメータ1〉

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分だけ負の方向に，さらにY軸に沿って〈パラメータ1〉×スケール値分負の方向に移動した点まで直線を描画します．

G〈パラメータ1〉

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分だけ負の方向に，さらにY軸に沿って〈パラメータ1〉×スケール値分正の方向に移動した点まで直線を描画します．

H〈パラメータ1〉

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分だけ正の方向に，さらにY軸に沿って〈パラメータ1〉×スケール値分正の方向に移動した点まで直線を描画します．

L〈パラメータ1〉

LPから，現座標系のX軸に沿って〈パラメータ1〉×スケール値分だけ負の方向へ移動した点まで直線を描画します．

M〈パラメータ 1〉,〈パラメータ 2〉

LP から，ワールド座標系の X 座標＝〈パラメータ 1〉，Y 座標＝〈パラメータ 2〉の点まで直線を描画します．

P〔〈パラメータ 1〉〕

LP を含み，〈パラメータ 1〉で指定したパレット番号で描かれた境界線で囲まれた領域を，色またはタイルストリングでぬりつぶします．〈パラメータ 1〉が省略された場合には, C サブコマンドで指定されているパレット番号が採用されます．P サブコマンドを実行するときは, LP はウィンドウ内になければなりません．P サブコマンドによってぬりつぶしを行う際の, 色やタイルストリングの指定方法については, Z サブコマンドを参照してください．

Q〈パラメータ 1〉,〈パラメータ 2〉

LP から，現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分，Y 軸に沿って〈パラメータ 2〉×スケール値分それぞれ正の方向へ移動した点までを対角線とする長方形を描画します．

R〈パラメータ 1〉

LP から，現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分正の方向に移動した点まで直線を描画します．

S〈パラメータ 1〉

グラフィック描画サブコマンドで指定される相対位置のパラメータのスケール値(倍率)を規定します．〈パラメータ 1〉はスケール値を表し，0 より大きな値でなければなりません．

初期状態では，〈パラメータ 1〉の値は 1 となっています．

T〈パラメータ 1〉,〈パラメータ 2〉

Q サブコマンドと同様にして長方形を描画した後，その内部をぬりつぶします．T サブコマンドによってぬりつぶしを行う際の, 色やタイルストリングの指定方法については, Z サブコマンドを参照してください．

U〈パラメータ 1〉

LP から, 現座標系の Y 軸に沿って,〈パラメータ 1〉×スケール値分負の方向に移動した点まで直線を描画します．

W〈パラメータ 1〉,〈パラメータ 2〉

LP から，現座標系の X 軸に沿って〈パラメータ 1〉×スケール値分，Y 軸に沿って〈パラメータ 2〉×スケール値分それぞれ正の方向へ移動した点まで直線を描画します．

X 〈パラメータ 1〉

〈パラメータ 1〉で指定される文字変数内のグラフィック描画サブコマンド列に従い図形を描画します．文字列変数内に，再びXサブコマンドを含めることも可能です．なお，〈パラメータ 1〉には，必ず"=〈文字変数〉；"の形を使用してください．

コンパイラにおいてはこの機能は使用できません．

Y 〈パラメータ 1〉

D，E，F，G，H，L，M，Q，R，UおよびWサブコマンドで描画する直線または長方形のラインスタイルを規定します．〈パラメータ 1〉は，&H0～&HFFFFの範囲の整数でなければなりません．ラインスタイルの内容については，LINEを参照してください．初期状態では，〈パラメータ 1〉の値は&HFFFFとなっています．

Z 〈パラメータ 1〉

TおよびPサブコマンドでぬりつぶす際の，ぬりつぶしの色またはタイルストリングによる模様を規定します．〈パラメータ 1〉は，パレット番号またはタイルストリングが格納されている文字列変数でなければなりません．ただし，タイルストリングを指定する場合は，必ず"=〈文字変数〉；"の形を使用してください．タイルストリングについては，〔2〕PAINTを参照してください．
初期状態では，Cサブコマンドにより規定されるパレット番号(Cサブコマンドを実行していないときはフォアグラウンドカラー)となっています．

D，E，F，G，H，L，M，Q，R，T，UおよびWサブコマンドの実行では，修飾子としてNが指定されていない限り，LPは指定された点に移ります．A，C，P，S，YおよびZサブコマンドの実行ではLPは変化しません．

また，A，C，S，YおよびZサブコマンドの指定は，指定のあるDRAWだけでなく，以降のDRAWにもその指定が引き継がれます．これらの指定は，SCREENを実行すると初期状態にもどります．

参照：LINE，〔2〕PAINT，サンプルプログラム17

## DSKF 関数 C

| | |
|---|---|
| 機 能 | ディスクの空き領域のサイズを得ます． |
| 書 式 | DSKF(〈ドライブ名〉) |
| 文 例 | PRINT DSKF("A：") |

〈ドライブ名〉で指定されたドライブにセットされているディスクの，未使用（空き）領域のサイズをバイト数で得ます．

# EDIT

| | |
|---|---|
| 機 能 | 指定された行を画面上に表示し，以降 ROLL UP キー，ROLL DOWN キーなどによるプログラム編集を可能にします． |
| 書 式 | EDIT 〈行番号〉 |
| 文 例 | EDIT 30<br>EDIT . |

EDIT を実行すると〈行番号〉で指定された行が表示され，カーソルは行の先頭に移動します．以降，ROLL UP キー，ROLL DOWN キー，カーソルキーを始めとする，各種プログラム編集キーを使ったプログラムの編集が可能になります．

〈行番号〉にピリオド（．）を指定すると BASIC インタプリタ内のポインタが示している現在行を指定したことになります．実際には，LIST で最後に表示された行や，編集機能で最後に修正された行が現在行となります．

コンパイラにおいてはこの機能は使用できません．

注意：P オプションつきでセーブされたファイルをロードして EDIT により，内容を表示，変更しようとした場合には，"Illegal　function　call" エラーとなります．

# END　C

| | |
|---|---|
| 機 能 | プログラムの終了を宣言します． |
| 書 式 | END |
| 文 例 | END |

プログラムの実行を終了し，すべてのファイルをクローズしてコマンドレベルにもどります．

コンパイラによりコンパイルされたプログラムの場合，MS-DOS に制御がもどります．

END はプログラムの実行を終了させたい所に置けばよく，また，いくつ置いてもかまいません．

**注意**：プログラムの最後のENDは省略することができますが，この場合，ファイルのクローズは行われません．

> コンパイラによりコンパイルされたプログラムの場合は，ENDを省略してもプログラムが終了すればファイルのクローズが行われます．

**参照**：CLOSE，STOP

# ENVIRON

**機　能**　BASICの環境文字列テーブル中の環境文字列のセット／更新を行います．

**書　式**　ENVIRON　〈文字列〉

**文　例**　ENVIRON　"PATH=¥BASIC"
ENVIRON　"STRING=ABCDEFG"

環境文字列テーブルはMS-DOS下で動作するすべてのプログラムに付与されるもので，親から子のプログラムへと引き継がれていきます．たとえば，MS-DOS下において，MS-DOSのSETコマンドあるいはPATHコマンドで設定された環境文字列テーブルは，BASICを起動した後も引き継がれます．また，BASICからCHILDでチャイルドプロセスを起動した場合も，チャイルドプロセスに引き継がれます．

ENVIRONはBASICの環境文字列テーブルに新しい環境文字列をセットしたり更新したりする命令ですが，環境文字列テーブルの内容はBASICに対しては何の働きもしません．チャイルドプロセスに引き継がれて初めて有効になります．

環境文字列テーブルは"名前=パラメータ"を基本単位とする文字列の集まりです．ENVIRONに指定する〈文字列〉は次のような形式で指定します．

(1) "名前=パラメータ"の形式で指定しますが，"名前=パラメータ"は1個しか指定できません．名前およびパラメータは1バイト系の英大文字，数字，記号からなる任意の文字列です．
(2) "名前=；"あるいは"名前="と指定された場合，指定された"名前"とパラメータは環境文字列から削除されます．ただし，"名前"が"PATH"の場合は，"PATH="が環境文字列テーブルに残されます．
(3) 指定された"名前"がすでに環境文字列テーブルに存在している場合，"パラメータ"の置き換えが行われます．この場合更新された〈文字列〉は，環境文字列テーブルの最後に移動します．
(4) 指定された"名前"が環境文字列テーブルに存在しない場合，その"名前"と"パラメータ"は，環境文字列テーブルの最後に追加されます．

環境文字列テーブル内の名前のうち"PATH"だけは特別の意味をもちます."PATH"に続く文字列は，チャイルドプロセスが起動される際に，そのチャイルドプロセス（プログラム）の格納されているディレクトリ名として参照されます.

CHILDにおいては，チャイルドプロセスのプログラム名を指定する際，ドライブ名とプログラム名（ファイル名）しか指定できません．この場合には，カレントディレクトリのみが，指定されたプログラムの探索対象となってしまいます.

このようなときに，"PATH"にディレクトリ名を設定しておくと，起動しようとするチャイルドプロセスがカレントディレクトリにない場合でも，システムが自動的にそのディレクトリ内を参照し，目的のプログラムをロードしてくれます.

例 1) CHILD "A：SUB1"
　　…ドライブAのディスクのカレントディレクトリ下のSUB1が起動される.

例 2) ENVIRON "PATH=¥BASIC"
　　CHILD "A：SUB1"
　　…ドライブAのディスクのBASICというディレクトリ下のSUB1が起動される.

例 3) ENVIRON "PATH=¥BASIC¥BASIC1"
　　CHILD "SUB1"
　　…カレントドライブのディスクのBASICというディレクトリ下に位置するディレクトリBASIC1に登録されているSUB1が起動される.

環境文字列テーブルの"PATH"以外の任意の名前は，チャイルドプロセスが独自に意味づけを行うものです．したがって，起動しようとするチャイルドプロセスが，環境文字列テーブルの内容を利用するように作成されていなければ，何をセットしても意味がありません.

注意：ENVIRONでは，BASICが起動される直前（コンパイラによってコンパイルされたプログラムの場合，そのプログラムが起動される直前）に作られた環境文字テーブルのサイズを超えてパラメータの設定を行うことはできません．したがって，ENVIRONを実行するためには，事前に必要な分だけの環境文字列テーブルのサイズを確保しておかなければなりません.

環境文字列テーブルのサイズを確保するには，BASICあるいはコンパイルされたプログラムの起動前に，MS-DOSのPATHあるいはSETコマンドで環境文字列テーブルに文字列をセットすることによって行います.

環境文字列テーブルのサイズは16バイト単位となっており，MS-DOSのPATHあるいはSETで環境文字列をセットすると，セットした文字列のバイト数を超える最小の16の倍数バイトが確保されます.

コンパイルされたプログラムの場合，プログラム中のRUNでファイル名の指定を行っているとそのファイル名についても環境文字列テーブル内のPATHが参照されます.

参照：CHILD，ENVIRON$，RUN

## ENVIRON$ 関数 C

機　能　BASIC の環境文字列テーブル内に設定されている環境文字列を取り出します．

書　式　1）ENVIRON$（<名前>）

　　　　2）ENVIRON$（<番号>）

文　例　PRINT　ENVIRON$（"PATH"）

　　　　A$＝ENVIRON$（"PARAM"）

　　　　PRINT　ENVIRON$(3)

書式 1）環境文字列テーブル内にセットされている環境文字列（"名前＝パラメータ"の形式）から，指定した〈名前〉に一致する環境文字列を捜し出し，それに設定されているパラメータを取り出します．

例）環境文字列テーブル中に"EDITOR＝JMAC"という環境文字列がセットされているときに，

A$＝ENVIRON$（"EDITOR"）

とすると，A$には"JMAC"が代入される．

書式 2）環境文字列テーブル内にセットされている環境文字列から，指定した〈番号〉番目に位置する環境文字列をそのまま（"名前＝パラメータ"の形式）取り出します．

環境文字列テーブルの詳細に関しては，ENVIRON を参照してください．

参照：ENVIRON

## EOF 関 数 C

機　能　ファイルの終了コードを調べます．

書　式　EOF(<ファイル番号>)

文　例　IF　EOF(1)　THEN　CLOSE　1：END

〈ファイル番号〉で指定されたファイルが終わりに達したかどうかを調べる関数です．終わりに達していれば－1(真)，そうでなければ 0(偽)を返します．〈ファイル番号〉で指定されたファイルは，入力モードでオープンされていなければなりません．

指定されたファイルがRS-232C回線(COM〈回線番号〉：)であった場合には，EOFは，バッファが空であったときに値を真とします.

参照：LOC，LOF，サンプルプログラム28

## ERASE　C

機能　配列変数を消去します.

書式　ERASE　〈配列変数名〉〔，〈配列変数名〉…〕

文例　ERASE　C，D$

〈配列変数名〉には，消去したい配列の変数名を指定します．添字や()を指定する必要はありません.

ERASEを実行すると指定した配列変数が消去されますから，元の配列変数に割り当てられていた分だけメモリ領域が増加します．したがって，同一変数名の配列変数をDIMで新たに宣言したり，他の目的に使用することもできるようになります.

注意：ERASEを実行せずに同一変数名で配列変数を宣言しようとすると，〝Duplicate Definition〞エラーが起こります.

参照：DIM，OPTION，BASE，サンプルプログラム5

## ERL／ERR　関数　C

機能　エラーの発生した行番号および発生したエラーのエラーコードを保持しています.

書式　1）ERL
　　　2）ERR

文例　IF　ERL=100　THEN　CLS：RESUME
　　　IF　ERR=7　THEN　STOP

エラーが発生した時点で，ERLはエラーの発生した行番号を，ERRはエラーコードを保持しています.

一般に，ERRおよびERLはON　ERROR　GOTOによって指定したエラー処理ルーチンにおいて，独自のエラー処理を行うために使用します.

参照：ERROR，ON　ERROR　GOTO，付録A，サンプルプログラム22

# ERROR

C

| 機 能 | エラー発生のシミュレート，エラーコードの利用者定義を行います． |
|---|---|
| 書 式 | ERROR 〈整数表記〉 |
| 文 例 | ERROR 200<br>ERROR A% |

〈整数表記〉には，0から255までの範囲の数を指定します．

この値が，すでにBASICでエラーコードとして使われている場合は，エラーメッセージを表示するとともに，プログラムの実行を停止します．エラーコードが定義されていない場合は，プログラムの実行を停止します．

どちらの場合も，エラー発生時のプログラム行番号はERLに，エラーコードはERRにセットされます．したがって，ON ERROR GOTOによるエラー処理ルーチンを利用して，独自のエラー処理を行うことができます．

参照：ERL／ERR，ON ERROR GOTO，付録A.エラーメッセージとその対策，サンプルプログラム22

# EXP 関数

C

| 機 能 | e(自然対数の底)に対する指数関数の値を得ます． |
|---|---|
| 書 式 | EXP(〈数式〉) |
| 文 例 | E=EXP(1)<br>PRINT EXP(A／2) |

〈数式〉の，eに対する指数演算の結果を値として得ます．

〈数式〉に倍精度実数が含まれる場合，得られる値は倍精度となりますが，他の場合には単精度となります．

# FIELD

Ⓒ

| 機能 | ランダムファイルバッファにフィールド変数を割り当てます. |
|---|---|
| 書式 | FIELD 〔#〕〈ファイル番号〉, 〈フィールド幅〉 AS 〈文字変数〉〔, 〈フィールド幅〉 AS 〈文字変数〉…〕 |
| 文例 | FIELD #1, 128 AS A$, 64 AS B$, 64 AS C$ |

FIELDは, OPENで設定されたランダムファイルバッファに対して, 入出力を行うためのフィールドの名前, およびその長さを割り当てます. したがって, FIELDの実行にさきだって, OPENによりファイルをランダムモードでオープンしておかなくてはなりません.

〈文字変数〉には, 割り当てたいフィールドの名前を文字型の変数名として指定します. この変数は, バッファにデータを代入したり, 代入したデータを参照するのに用いられ, フィールド変数と呼ばれます.

〈ファイル番号〉は, OPENによってファイルバッファをオープンしたときに指定した番号です.

〈フィールド幅〉は, 〈文字変数〉で指定したフィールド変数に割り当てる文字数で, 0～255の範囲のバイト数で指定します. 1つのバッファには, 文字数合計がインタプリタ起動時に／Sスイッチで指定したランダムファイルのレコードサイズまでならば, いくつのフィールド変数を割り当ててもかまいません.

コンパイラの場合, ディスクファイルアクセス用バッファサイズはREM $FILEで宣言します.

フィールド変数へ代入する(LSETあるいはRSETで行う)データの長さは, フィールド幅を超えてはなりません.

数値データをフィールドにセットする場合, 数値データの型に応じてMKI$, MKS$, MKD$の各関数を用いて文字型化してからLSET／RSETを行います. このとき, 整数型の場合は2バイト長, 単精度実数型の場合は4バイト長, 倍精度実数型の場合は8バイト長にそれぞれ変換されますので, 〈フィールド幅〉もこれに合わせて設定する必要があります.

2バイト系日本語文字列をフィールドにセットする場合, 日本語1文字あたり2バイト分のフィールドを必要とします. 〈フィールド幅〉の設定を行うときにはこのことを考慮するようにしてください. たとえば,

```
LSET A$="日本語"
```

のためには最低6バイト分のフィールドを必要としますので, フィールド幅には6以上を設定しなくてはなりません.

なお, 1つのバッファに対して, 異なった形式で複数のFIELDを実行してもかまいません.

**注意：** 各フィールド変数への値の代入を通常の代入文あるいはINPUTなどで行わないでください．もし行うと，その変数は一般の文字変数領域に登録されてしまうため，FIELDでのバッファ割り当てが無効となり，正しいファイルの入出力が行われなくなります．

**参照：** GET，LSET／RSET，MKI$／MKS$／MKD$，OPEN，PUT，サンプルプログラム1，2，26，27，29

# FILES／LFILES　Ⓒ

**機　能**　ディスクに入っているファイルの名前，サイズ，作成日付などの一覧を出力します．

**書　式**　1）FILES 〔<ファイルディスクリプタ>〕
2）LFILES 〔<ファイルディスクリプタ>〕

**文　例**
```
FILES
LFILES "B:"
FILES "DOCUMENT.TXT"
LFILES "A:¥SOURCE¥*.BAS"
```

〈ファイルディスクリプタ〉で指定したドライブ，ディレクトリ，ファイルに関する情報（ファイル名またはディレクトリ名，ファイルのサイズ，作成日付などの一覧）を，書式1）は画面に，書式2）はプリンタに，それぞれ出力します．

〈ファイルディスクリプタ〉は次のような形で指定します．

〔〈ドライブ名：〉〕〔¥〕〔〈ディレクトリ名〉¥〕…〔〈ディレクトリ名〉¥〕〔〈ファイル名〉〕〔.〈拡張子〉〕

● 〈ファイルディスクリプタ〉全体が省略された場合

カレントドライブにあるディスクの，カレントディレクトリの情報が出力されます．

例）　FILES

● 〈ドライブ名：〉のみ指定された場合

指定ドライブにあるディスクの，カレントディレクトリの情報が出力されます．

例）　FILES "B:"

Bドライブにあるディスクの，カレントディレクトリの情報が出力されます．

● 〈ディレクトリ名〉のみ指定された場合

カレントドライブにあるディスクの，指定ディレクトリの情報が出力されます．このとき，¥から始まるディレクトリ名を指定したときはルートディレクトリからの指定（絶対指定）とな

り，頭に¥をつけないでディレクトリ名を指定したときはカレントディレクトリからの指定(相対指定）となります．

ディレクトリが階層化されている場合は，複数のディレクトリ名を¥でつなぐことにより，目的のディレクトリを指定することができます．

例） FILES "¥SOURCE¥BASIC"

カレントドライブにあるディスクの，ルートディレクトリ内のSOURCEディレクトリのさらに下にあるBASICディレクトリの情報が出力されます．

FILES "DOC"

カレントドライブ内にあるディスクの，DOCディレクトリの情報が出力されます．

**●〈ドライブ名：〉，〈ディレクトリ名〉，〈ファイル名〉〔．〈拡張子〉〕が組み合わされて指定された場合**

指定ドライブにあるディスクの指定ディレクトリ内の，指定ファイルについての情報が出力されます．

〈ディレクトリ名〉については，絶対指定／相対指定のいずれも可能です．この場合，〈ドライブ名：〉が省略されたときにはカレントドライブが，〈ディレクトリ名〉が省略されたときにはカレントディレクトリが，それぞれ指定されたものとみなされます．

〈ファイル名〉〔．〈拡張子〉〕については，ワイルドカード（＊，？）を使用することもできます．ワイルドカード指定を行わなかった場合は，そのファイルがバイナリ形式によってセーブされたファイルであればファイル名と拡張子との間にピリオド（.）が表示され，その他のファイルであればファイル名と拡張子との間いはスペース（空白）となります．

例） FILES "B：¥TEXT¥＊.DOC"

Bドライブにあるディスクの，ルートディレクトリ内のTEXTディレクトリ内にある，拡張子がDOCであるファイルすべてについての情報が出力されます．

FILES "A：BAS¥TEST.BAS"

Aドライブにあるディスクの，カレントディレクトリのさらに下にあるBASディレクトリ内に収められているTEST.BASファイルについての情報が出力されます．この場合は，TEST.BASファイルがバイナリ形式でセーブされたものであるか否かが，ピリオドの有無によって表示されます．

## FIX 関数 C

| | |
|---|---|
| 機能 | 数値の整数部を得ます． |
| 書式 | FIX(〈数式〉) |
| 文例 | F＝FIX(B／3) |

〈数式〉の値の小数点以下を取り去った値を得ます.

FIX と INT の違いは,〈数式〉の値が負のときに, INT は〈数式〉の値を超えない最大の整数を返すのに対し, FIX はたんに小数点部分だけを取り去った値を返すことです.

参照：INT

# FOR…TO…STEP～NEXT C

機能　FOR から NEXT までの区間中にある一連の命令を, 繰り返して実行します.

書式
```
FOR 〈変数名〉=〈初期値〉 TO 〈終値〉〔STEP 〈増分〉〕
 ∫
NEXT 〔〈変数名〉〕〔,〈変数名〉〕
```

文例
```
FOR J=0 TO 100 STEP 2
FOR K=10 TO 0 STEP -1
 ∫
NEXT K, J
```

FOR～NEXT ループ中(FOR と NEXT の間)に置かれた命令を, FOR 中で指定した条件に従って繰り返して実行します.

〈変数名〉で指定される変数(ループ変数と呼ぶ)は, 整数型または単精度型でなくてはなりません.

〈初期値〉には, 変数の初期値を設定します.〈終値〉には, 変数の最終値を設定します.〈増分〉には, 初期値と最終値との間の増分を指定します.

文例を例にとると, はじめ J=0 が設定され, FOR 以降の命令を実行します. プログラムの実行が NEXT まで来ると, J の値が増分の 2 だけ増やされて J=2 となり, 再び, FOR 以下が実行されます. J=100 となるまで FOR と NEXT の間の処理が繰り返されます.

STEP 〈増分〉が省略された場合には,〈増分〉は 1 とみなされます.

次の場合には, FOR～NEXT は実行されずに, NEXT の次へ実行が移ります.

(1)〈増分〉が正の値で,〈初期値〉が〈終値〉より大きい場合.

(2)〈増分〉が負の値で,〈初期値〉が〈終値〉より小さい場合.

ただし,〈変数〉には〈初期値〉が代入されます.

1 つの FOR～NEXT の中にはもう 1 つの FOR～NEXT を置くことができます(入れ子構造, ネスト構造などと呼ぶ). この場合, それぞれの〈変数名〉には別のものを使わなければなりません. また, このとき, 1 つの FOR～NEXT は完全に他の FOR～NEXT の内部になければなりません.

注意：FOR と NEXT は必ず1対1に対応していなければなりません．
　　　また，FOR〜NEXT ループ内へ外部から GOTO などでジャンプして入ってきたり，逆にループ内から外部へジャンプしたりするようなプログラムは，その動作が保証されなくなります．

参照：WHILE〜WEND，サンプルプログラム6

## FRE 関数 C

機　能　各種制御領域の全体サイズ，未使用領域サイズを得ます．

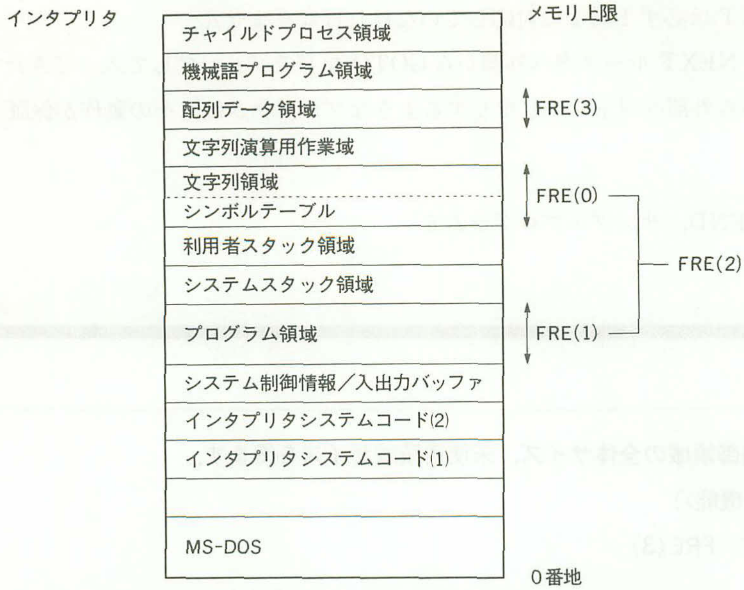
書　式　FRE(〈機能〉)

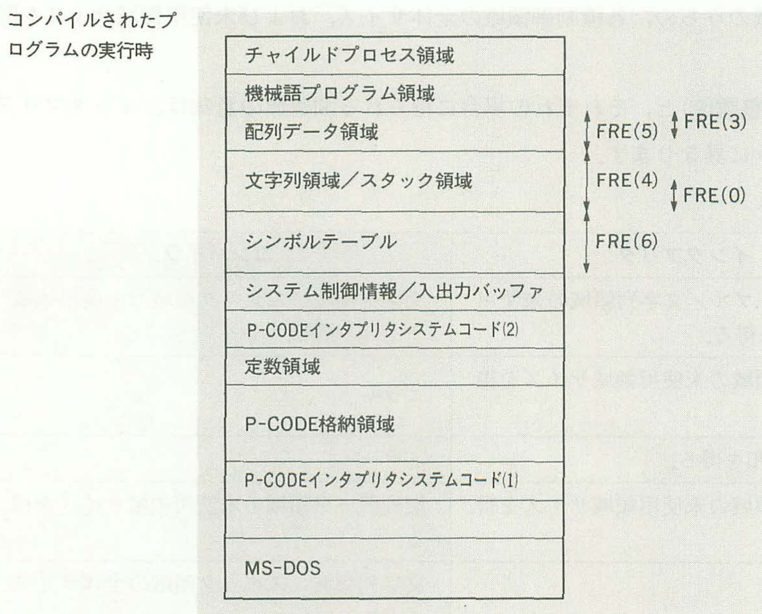
文　例　PRINT　FRE(3)

利用可能なメモリ領域のうちの，各種制御領域の全体サイズ，および未使用領域サイズを関数値として得ます．

〈機能〉に指定する値(整数値)と，それぞれの場合に得られる関数値の意味は，インタプリタとコンパイラで次のように異なります．

| 機能 | インタプリタ | コンパイラ |
|---|---|---|
| 0 | シンボルテーブル／文字列領域の未使用領域サイズを得る． | 文字列領域／スタック領域の未使用領域サイズを得る． |
| 1 | プログラム領域の未使用領域サイズを得る． | エラー |
| 2 | 機能0，1の和を得る． | エラー |
| 3 | 配列データ領域の未使用領域サイズを得る． | 配列データ領域の未使用領域サイズを得る． |
| 4 | エラー | 文字列領域／スタック領域の全体サイズを得る． |
| 5 | エラー | 配列データ領域の全体サイズを得る． |
| 6 | エラー | シンボルテーブル＋システム制御情報／入出力バッファの全体サイズを得る． |




参照：CLEAR，SEGPTR

# GET

Ⓒ

機　能　ファイル中のデータをファイルバッファに読み込みます．

書　式　GET　〔#〕〈ファイル番号〉〔,〈数値〉〕

文　例　GET　#3, 5

GET　1

〈ファイル番号〉で指定されたファイル中のデータを，対応するバッファ中に読み込みます．

GET は，指定されたファイルがディスクファイルか，キーボードファイルかによってその動作が異なります．

(1) ディスクファイルの場合

ランダムファイルからのデータをランダムファイルバッファに読み込みます．指定されたディスクファイルはランダムモードでオープンされていなければなりません．

〈数値〉はファイルのレコード番号として解釈され，指定されたレコードがバッファに読み込まれます．レコード番号の最小値は 1，最大値は 65535 です．〈数値〉が省略された場合には，直前に行われた，GET，PUT で指定されたレコードの次のレコードが読み込まれます．

なお，GET を行った後で，INPUT #，LINE　INPUT #などのシーケンシャル・アクセスを行った場合には，バッファに入っているデータから入力が行われますので注意してください．

(2) キーボードファイル(KYBD：)の場合

キーボードから入力した文字をバッファ中に読み込みます．キーボードファイルは入力モードでオープンされていなければなりません．

〈数値〉は，キーボードからバッファに読み込む文字(バイト)数と解釈されます．0 から 255 までの値で指定します．〈数値〉が省略されたとき，および 0 が指定されたときには 256 文字を読み込みます．

GET はバッファへの読み込みを行うものです．読み込んだデータは FIELD で指定した変数(フィールド変数)で参照することができます．

参照：FIELD，OPEN，PUT，サンプルプログラム 2，27，29

# GET@ C

**機　能**　画面上のグラフィックパターンを配列変数に読み込みます．

**書　式**　GET〔@〕 (Sx1, Sy1)－ { (Sx2, Sy2) | STEP(x, y) } ，〈配列変数名〉〔(〈添字〉)〕

**文　例**　GET　(100, 50)－(130, 70), G%

画面上のスクリーン座標の 2 点(Sx1, Sy1)と(Sx2, Sy2)を対角点とする四角形のグラフィックパターンを，〈配列変数名〉で指定された配列変数に読み込みます．

座標の指定は，ビューポート内に展開されるスクリーン座標を使用します(PUT@も同様)．2 つ目の座標指定には，STEP を使って相対座標による指定もできます．

〈配列変数名〉は，グラフィックデータを読み込むために用意する数値型の配列変数の名前です．

〈添字〉は，グラフィックパターンを配列変数に読み込む際に，どの要素から格納し始めるかを指定するものです．省略した場合は配列の最初から格納します．〈添字〉を使うことにより，1つの配列変数に対して複数のグラフィックパターンを格納することができます．

GET@を実行する前に，DIMで配列変数の大きさを宣言しておかなければなりません．確保すべき配列変数の大きさは，1つの配列に1つのパターンしか読み込まない場合，次の計算式で求めることができます．複数パターンの場合は，それぞれのパターンについて計算し，その合計分を確保します．

**必要なバイト数＝((横のドット数＋7)¥8)＊縦のドット数＊M＋4**

ただし，白黒モードのとき ―― M＝1

8色カラーモードのとき ―― M＝3

16色カラーモードのとき ―― M＝4

**添字の値＝必要なバイト数¥N＋1**

Nは配列変数の型によって変わります．

整数型配列 ―― N＝2

単精度型配列 ―― N＝4

倍精度型配列 ―― N＝8

この計算式によって求めた〝添字の値〟を，DIMで配列変数を宣言するときに〈添字の最大値〉(DIM参照)として指定します．この計算式で求められる添字の値は，1次元配列で添字の最小値が0のとき(OPTION　BASE参照)のものです．一般にGET@では1次元の配列変数を用います．

**注意**：このGET@はPUT@と密接な関係にあり，通常ペアで使用されます．また，GET@実行後，LP(最終参照点)は(Sx2，Sy2)に移されます．

**参照**：DIM，OPTION　BASE，PUT@，サンプルプログラム18

# GOSUB　Ⓒ

| | |
|---|---|
| 機　能 | サブルーチンを呼び出します． |
| 書　式 | GOSUB　〈行番号〉 |
| 文　例 | GOSUB　＊SUB1<br>GOSUB　550 |

サブルーチンとは，他から独立したプログラムで，RETURNで終っているものをいいます．GOSUBは，指定した〈行番号〉から始まるサブルーチンを呼び出します．

なお，〈行番号〉の代わりにラベル名を使うこともできます．

1つのサブルーチンの中から他のサブルーチンを呼び出すこともできます．これを，サブルーチンの多重化と呼びます．この多重化は，メモリのスタック領域(CLEAR 参照)の容量が許す限り行うことができます．

注意：サブルーチンを多重化してスタック領域が足りなくなると，"Out of memory"エラーが発生します．

参照：CLEAR，RETURN，サンプルプログラム7，30，31，33

# GOTO／GO TO　C

機　能　指定された行へジャンプし，そこからプログラムを実行します．

書　式　1) GOTO 〈行番号〉

2) GO TO 〈行番号〉

文　例　GOTO 500

GOTO *EXIT

〈行番号〉の行へジャンプして，プログラムの実行を継続します．

〈行番号〉の代わりにラベル名を使うこともできます．

注意："GO" と "TO" の間にスペース1個を入れても，まったく同じ意味です．しかし2個以上の場合には，GOTO とは解釈されません．

参照：GOSUB，RETURN，RUN，サンプルプログラム7，28，29，30

# HELP ON／OFF／STOP　C

機　能　HELP キーによる割り込みの許可，禁止，停止を制御します．

書　式　1) HELP ON

2) HELP OFF

3) HELP STOP

文　例　HELP OFF

HELP キーを押したことによる割り込みの許可，禁止，停止を宣言します．

書式1) 割り込みを許可します．以後 HELP キーを押すごとに割り込みが発生し，ON HELP

GOSUBによって定義された処理ルーチンに分岐します.

書式2) 割り込みを禁止します. 以後 HELP キーを押しても処理ルーチンへの分岐は起こりません( HELP キーは通常の動作になります).

書式3) 割り込みを停止します. 以後 HELP キーを押しても押されたことを覚えているだけで, 処理ルーチンへの分岐は起こりません. しかし, その後, HELP ON によって割り込みが許可されると, 前に HELP キーを押したことによって, 処理ルーチンに分岐します.

注意:プログラムの終了時にはHELP OFFを実行しておいてください. さもないと本来の HELP キーの機能が失われます.

参照:ON HELP GOSUB

# HEX$ 関 数 C

| | |
|---|---|
| 機 能 | 10進数を16進数に変換し, その文字列を得ます. |
| 書 式 | HEX$(〈数式〉) |
| 文 例 | A$=HEX$(X)<br>PRINT HEX$(255) |

〈数式〉の値を16進数に変換して, その文字列を得る関数です. 〈数式〉の値は, −32768から32767(または0から65535)までの範囲になければなりません. 〈数式〉の値と得られる文字列との対応は次のとおりです.

| 〈数式〉の値 | | 得られる16進表記文字列 |
|---|---|---|
| 0〜32767 | : | 0〜7FFF |
| −32768〜−1 | : | 8000〜FFFF |
| 32768〜65535 | : | 8000〜FFFF |

参照:OCT$, STR$, VAL, サンプルプログラム24

# IEEE 関数 G C

| | |
|---|---|
| 機 能 | GP-IBの諸状態を調べます. |
| 書 式 | IEEE (〈機能〉) |
| 文 例 | A=IEEE (0) |

GP-IB(IEEE-488)のいろいろな状態を調べる関数です。

〈機能〉には0～2，4～7の数値を指定します（3は指定できません）。各数値を与えたときに得られる値と，その意味を次に説明します。

IEEE(0)　CMD　DELIMで指定されたデリミタを調べます。得られる値とデリミタの関係は次のとおりです。

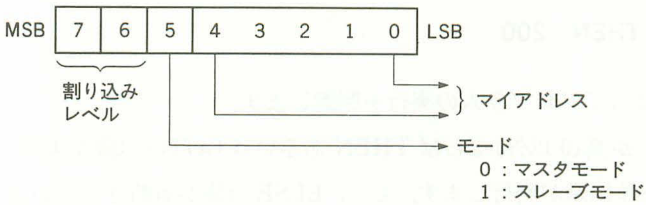
0　：　$C_R + L_F$

1　：　$C_R$

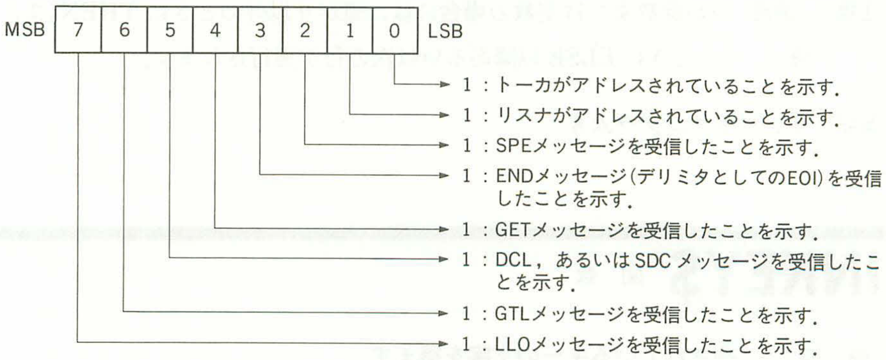
2　：　$L_F$

3　：　EOI

IEEE(1)　PC-9801-29Nのイニシャライズ状態を調べます。得られる値は次のような8ビットのデータです。




IEEE(2)　PC-9800-29Nのトーカ，リスナの状態および受信したインタフェースメッセージを調べます。得られる値は次のような8ビットのデータで，この関数が呼ばれた直前の状態がわかります。




IEEE(4)　シリアルポール時にSRQ（サービスリクエスト）を送出している装置のデバイスステータスを調べます。

IEEE(5)　シリアルポール時にSRQを送出している装置のトーカアドレスを調べます。

IEEE(6)　シリアルポール時にシリアルポールに応答しない装置があった場合，そのトーカアドレスを調べます。

IEEE(7)　　パラレルポールを行い，パラレルポールによって得たデータバイトを調べます．

参照：STATUS

## IF…THEN～ELSE／IF…GOTO～ELSE　C

| 機　能 | 論理式の条件判断を行います． |
|---|---|
| 書　式 | 1) IF 〈論理式〉 THEN ｜〈文〉／〈行番号〉｜ 〔ELSE ｜〈文〉／〈行番号〉｜〕<br>2) IF 〈論理式〉 GOTO 〈行番号〉 〔ELSE ｜〈文〉／〈行番号〉｜〕 |
| 文　例 | IF A$="Y" THEN ＊START ELSE ＊EXIT<br>IF SS THEN 200 |

〈論理式〉の条件によってプログラムの実行を制御します．

すなわち，〈論理式〉が真(0以外)ならばTHENあるいはGOTO以降を実行します．〈論理式〉が偽(0)ならばELSE以降が実行します．もし，ELSE以降が省略されている場合には，次の行が実行されます．

IF…THEN(GOTO)～ELSEにおいて，ELSEに続けて別のIFを置いて多重にすることができます．ただし，多重にできるのは1行(255バイト)に書ける範囲に限られます．

注意：〈論理式〉が変数または定数の場合には，値が0以外のときにTHEN(またはGOTO)以降が，0のときにELSE以降あるいは次の行が実行されます．

参照：サンプルプログラム8

## INKEY$　関数　C

| 機　能 | 押されているキーの文字を得ます． |
|---|---|
| 書　式 | INKEY$ |
| 文　例 | IF INKEY$="" THEN ＊WAIT |

押されているキーの文字(1文字)を取り出します．もしキーが押されていなければINKEY$の値はヌルストリング(空の文字列)となります．

キーボードバッファにすでに入力済みの文字が入っている場合には，バッファの先頭の１文字を取り出します．

INKEY$では，押されたキーは画面上には表示されません．また，INKEY$は，INPUTやINPUT$などとは違い，キー入力待ちを行いません．

注意：CTRL＋C，STOPキーはINKEY$で取り出すことはできません．

参照：INPUT，INPUT$，KINPUT，LINE INPUT，サンプルプログラム29

## INP 関数 C

| | |
|---|---|
| 機能 | 入力ポートから値を読み取ります． |
| 書式 | INP(〈ポート番号〉) |
| 文例 | A＝INP(15) |

〈ポート番号〉で指定された入力ポートから８ビットのデータを読み取り，それを関数値とします．〈ポート番号〉には0～32767(&H0000～&H7FFF)の値を指定します．

〈ポート番号〉と装置の対応は，各機種本体に添付されているハードウェアマニュアルを参照してください．

参照：OUT

## INPUT C

| | |
|---|---|
| 機能 | キーボードから入力されたデータを，変数に代入します． |
| 書式 | INPUT 〔〈プロンプト文〉 { ; \| , } 〕〈変数〉〔,〈変数〉…〕 |
| 文例 | INPUT A<br>INPUT "住所"；B$ |

INPUTが実行されると，プログラムはキーボードからの入力待ちの状態となります．

〈プロンプト文〉には，データの入力を受ける前に画面に表示されるメッセージを指定します．〈プロンプト文〉にセミコロン(；)が続いた場合には，さらにクエッションマーク(？)と１桁のスペースが画面に表示されます．コンマ(,)が続いた場合には〈プロンプト文〉の後には何も表示されません．

〈変数〉には，キーボードから入力したデータが代入されます．〈変数〉をコンマ(,)で区切っ

て複数個指定した場合には，入力するデータもコンマで区切って〈変数〉の数だけ入力しなければなりません．また，対応する〈変数〉とデータの型とは一致していなければなりません．

何も入力しないでリターンキーだけを押した場合には，0やヌルストリング(空の文字列)が入力されたとみなされます．この方法で0やヌルストリングの入力ができますが，この場合も〈変数〉が複数の場合，必要数のコンマだけは入力しなければなりません．

文字変数に文字列を代入する場合，コンマや文字列の前後に意味のある空白を入力するには，ダブルクォーテーション(")で文字列を囲む必要があります．この場合ダブルクォーテーションは，文字として入力されません．これ以外の場合には，文字列をダブルクォーテーションで囲む必要はありません．

入力を中断したい場合，[STOP]キーあるいは[CTRL]＋[C]キーを入力します．プログラムの実行中に入力を中断すると，STOP ON の状態にない限り，プログラムの実行も中断され，コマンドモードにもどります．このとき，それまでに入力した値はすべて無効となります(変数には値が代入されません)．

日本語の入力を行う場合には，[CTRL]＋[XFER]を入力して日本語入力モードにします($N_{88}$-日本語BASIC(86)(MS-DOS版)ユーザーズマニュアル参照)．

**注意**：〈変数〉と代入する値の型が違っていた場合には"?Redo from start"と画面表示して再び入力待ちとなります．

また，割り込み機能を使用しているプログラム中で，INPUTの実行中に割り込み動作が行われた場合，割り込み処理ルーチンから復帰したときにINPUTの次の命令に実行が移ってしまいますので，注意が必要です．

**参照**：INPUT WAIT, KINPUT, LINE INPUT, STOP ON／OFF／STOP, サンプルプログラム1，7，8，9，22，28，29，39

# INPUT #　[C]

**機能**　シーケンシャルファイルからデータを読み込み，変数に代入します．

**書式**　INPUT #〈ファイル番号〉，〈変数名〉〔，〈変数名〉…〕

**文例**　INPUT #1，A，B

INPUT #2，NAME$

〈ファイル番号〉には，OPENによって，そのファイルをオープンしたときに指定した番号を使います．

〈変数名〉の型は，ファイルから読み込まれるデータの型と対応していなければなりません．

ファイルに書き込まれているデータは，それが数値変数に代入される値(数字の並び)ならば，

空白，コンマあるいはキャリッジリターン(CHR$(13))で区切られていなければなりません．また，データが文字変数に代入される値(任意の文字の並び)ならば，コンマあるいはキャリッジリターンで区切られていなければなりません．

参照：OPEN，サンプルプログラム28

# INPUT$ 関数 [C]

[機　能] 指定されたファイルより指定された長さの文字列を読み込みます．

[書　式] INPUT$(〈文字数〉〔,〔#〕〈ファイル番号〉〕)

[文　例] WORD$=INPUT$(6, #2)

〈ファイル番号〉には，OPEN によってそのファイルをオープンしたときに指定した番号を使います．〈ファイル番号〉が省略された場合には，キーボードからの入力を読み込みますが，INPUT と異なり入力された文字は画面には表示されません．

〈文字数〉には，読み込みたい文字列の長さをバイト数で指定します．

INPUT$は指定された〈文字数〉の文字が入力されるのを待ち続けますが，すでにバッファに入力済みのデータがある場合には，バッファ中から文字を読み込みます．

また，INPUT$は，[STOP]キー，[CTRL]＋[C]を除くすべての文字をそのまま読み込みますので，INPUT や LINE　INPUT では入力することのできないキャリッジリターン(CHR$(13))なども入力することができます．

参照：INKEY$，INPUT，LINE　INPUT，サンプルプログラム22，29

# INPUT@ [G] [C]

[機　能] 指定したトーカから送られてくるデータを受信して，変数に代入します．

[書　式] 1) INPUT@〔〈トーカアドレス〉〕〔,〔〈リスナアドレス〉〕〕；〈変数〉〔,〈変数〉…〕

2) INPUT@；〈変数〉〔,〈変数〉…〕

[文　例] INPUT@1, 3；A$, B$

INPUT@；A, C$

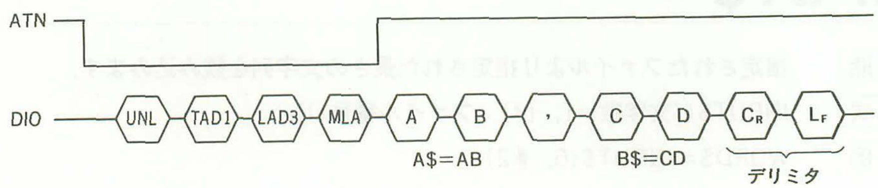
書式 1) マスタモードの場合に使用します．

ATN（アテンション）を true にして UNL（アンリスン）コマンド，トーカアドレス，リスナアドレス，MLA（マイリスンアドレス）を順次送出します．その後，ATN を

falseにして，指定されたトーカより送信されてくるデータを受信し，変数に代入します．

文字変数の場合，頭のスペースおよび終わりのスペースは無視され，さらに255バイトを超えると，超えた部分は次の文字変数に代入されます．複数個のデータを受信する場合は，変数をコンマ(,)で区切って記述します．この場合，","を受信するたびにデータの代入が行われます．

例）INPUT@　1，3；A$，B$を実行した場合のバスタイミング




INPUT@のタイミング

書式 2）スレーブモードの場合に使用します．

スレーブモードでは，アドレスの指定ができないため，アドレスは省略しなければなりません．この命令が実行されると，リスナがコントローラによりアドレスされるまで待ちます．MLA受信後，ATNがfalseになり，トーカからデータが送出されてくると，そのデータを受信して，変数に代入します．

注意：〈変数〉には文字変数，数値変数のいずれでも記述できますが，受信したデータと記述された型とは一致していなければなりません．

また，送られてきたデータの数と変数の数が一致しない場合，データの数の方が多いときは余分なデータは無視されます．変数の数の方が多いときは，余分な文字変数にはヌルストリング（空の文字列）が，数値変数には0が代入されます．

参照：LINE　INPUT@，PRINT@

# INPUT WAIT 　C

| | |
|---|---|
| 機　能 | キーボードから入力されたデータを，変数に代入します．その際，入力待ち時間を制限することができます． |
| 書　式 | INPUT　WAIT　〈待ち時間〉,〔〈プロンプト文〉 { ; / , } 〕〈変数名〉〔,〈変数名〉…〕 |
| 文　例 | INPUT　WAIT　100,"Your　name"；NA$<br>INPUT　WAIT　50, B$ |

INPUT WAIT は待ち時間に制限があることを除けば，INPUT と同様に機能します．

〈待ち時間〉で指定された時間だけキーボードからの入力を待ちます．〈待ち時間〉の単位は 0.1 秒です．入力が行われなかったときは，次の行へ実行が移ります．

注意：この文の後にマルチステートメントとして他の文を続けた場合，制限時間内に入力が行われたときは後の文が実行されますが，入力が行われなかったときは後の文は無視されて次の行へ実行が移りますから，注意してください．

参照：INPUT

## INSTR 関数 Ⓒ

機能　文字列の中から指定文字列を捜して，その文字の位置を得ます．

書式　INSTR(〔〈位置〉,〕〈文字列 1〉,〈文字列 2〉)

文例　J=INSTR(A$,"J")

PRINT　INSTR("ONETWOTHREE","TWO")

〈文字列 1〉の中から〈文字列 2〉が捜され，見つかればその位置が，見つからなければ 0 が，それぞれ値として得られます．

〈位置〉には，〈文字列 1〉中で捜し始める位置を文字数(整数値)で指定します．〈位置〉が省略された場合は 1 が指定されたものとみなされ，〈文字列 1〉の最初から捜し始めます．

〈文字列 2〉に""(ヌルストリング)を指定すると，INSTR の値は〈位置〉と同じになります．

参照：KINSTR

## INT 関数 Ⓒ

機能　小数点以下を切り捨てた整数値を得ます．

書式　INT(〈数式〉)

文例　PRINT　INT(1.123)

A=INT(RND * 255)

〈数式〉の値を超えない最大の整数を得ます．

INT と FIX の違いは，〈数式〉の値が負のときに，FIX はたんに小数点部分だけを取り去った値を返すのに対し，INT は〈数式〉の値を超えない最大の整数を返すことです．

参照：FIX，CINT

# IRESET REN　G C

機　能　REN（リモートイネーブル）を false にします.
書　式　IRESET　REN
文　例　IRESET　REN

REN メッセージを false で送出します.

注意：この命令は，マスターモードでのみ使用できます.
　PC-9801-29N は，REN ラインが true でも false でも常にリモート状態になっており，IEEE-IB を制御することができます.

参照：ISET　REN

# ISET IFC　G C

機　能　IFC（インタフェースクリア）の送出をします.
書　式　ISET　IFC〔,〈整数〉〕
文　例　ISET　IFC

IFC メッセージを true で送出した後，再び IFC メッセージを false にもどします.
〈整数〉は送出時間を一定時間以上保証するために指定するものです．実際に保証される時間は，〈整数〉×100$\mu$s 以上となります．省略すると 1 が採用され，100$\mu$s 以上 IFC を true にします.

注意：この命令は，マスターモードでのみ使用できます.

# ISET REN　G C

機　能　REN（リモートイネーブル）を true にします.
書　式　ISET　REN
文　例　ISET　REN

REN メッセージを true で送出します.

注意：この命令は，マスターモードでのみ使用できます.

参照：IRESET　REN

## ISET SRQ　G C

| 機能 | SRQ（サービスリクエスト）の送出を行います. |
|---|---|
| 書式 | ISET　SRQ〔@〕〔N〕 |
| 文例 | ISET　SRQ@ |

SRQ（サービスリクエスト）を true で送出します.

@が指定された場合は，デバイスステータスが送出されると同時に，EOI が true となります.

N の指定がない場合には，コントローラからシリアルポールされるまで待ちます．コントローラからシリアルポールが行われ，MTA（マイトークアドレス）が送出されてくると，ATN が false になったことを確認して SRQ を false にし，変数 STATUS に格納されているステータスバイトをコントローラに対して送出します.

注意：この命令は，スレーブモードでのみ使用できます.

## JIS$　関数　C

| 機能 | 2 バイト系日本語文字の漢字コードを得ます. |
|---|---|
| 書式 | JIS$(〈文字列〉) |
| 文例 | C$＝JIS$("漢字") |

〈文字列〉の最初の 2 バイトを 16 進表記 4 桁のキャラクタコードの文字列に変換します.

〈文字列〉の最初に 2 バイト系日本語文字を指定した場合には，JIS コードが得られます.

〈文字列〉の最初の 2 バイトが 1 バイト系文字 2 つの場合は，2 つの文字の 16 進数 2 桁のキャラクタコードが連続して得られます.

〈文字列〉が 2 バイト以下の場合は，"Illegal　function　call" エラーになります.

参照：ASC，CHR$，KNJ$，サンプルプログラム36

## KACNV$ 関数 C

機能　2バイト系全角文字を，対応する1バイト系の英数カナ文字に変換します．

書式　KACNV$(〈文字列〉)

文例　A$=KACNV$(B$)

PRINT　KACNV$("アイウＡＢＣ")

〈文字列〉中の2バイト系全角文字を1バイト系英数カナ文字に変換します．〈文字列〉中の1バイト系英数カナ文字には何も変換操作を行いません．

注意：日本語キャラクタモードでない場合と，〈文字列〉中に対応する1バイト系英数カナ文字のない2バイト系全角文字が含まれている場合は，"Illegal　function　call"エラーとなります．

参照：AKCNV$，サンプルプログラム35

## KEXT$ 関数 C

機能　文字列の中から1バイト系英数カナ文字だけ，あるいは，2バイト系日本語文字だけのどちらかを抜き出します．

書式　KEXT$(〈文字列〉，〈機能〉)

文例　A$=KEXT$(B$，0)

〈機能〉に指定する値により，特定のタイプの文字列だけを抜き出します．

0　：　1バイト系英数カナ文字だけを抜き出す．

1　：　2バイト系日本語文字だけを抜き出す．

〈機能〉で指定したタイプの文字がない場合，KEXT$の値はヌルストリング(空の文字列)となります．

参照：KLEN，KTYPE，サンプルプログラム37

# KEY　Ⓒ

機能　キーボードの上部にあるファンクションキーに文字列を定義します．

書式　KEY　<キー番号>，<文字列>

文例　KEY　1,"Yes"+CHR$(13)

KEY　3，CHR$(&H22)+"BASIC"+CHR$(&H22)

<キー番号>には，10個あるファンクションキーの番号を指定します．たとえば，f･1キーならば1を，f･2キーならば2を指定します．

<文字列>には，それぞれのキーに記憶させておく文字列を指定します．

この命令を実行すると，以後，各ファンクションキーを押すたびに，それぞれに定義した文字列を入力できます．

各ファンクションキーには，最大15バイトまでの文字列およびコントロール文字を定義できます．キーボードから直接入力できない文字は，CHR$関数を使って定義します．いちど定義したファンクションキーの内容は，新たに定義し直すかリセットするまで変わりません．

**注意**：ファンクションキーに2バイト系日本語文字列を定義することはできません．

参照：CHR$，KEY　LIST

# KEY LIST　Ⓒ

機能　ファンクションキーの内容を画面に表示します．

書式　KEY　LIST

文例　KEY　LIST

ファンクションキーの内容の一部は画面の最下行に，表示させておくことができますが，KEY　LISTを使えば，そのすべての内容を画面に表示させることができます．

参照：CONSOLE，KEY

# KEY ON／OFF／STOP　C

機能　ファンクションキーによる割り込みの許可，禁止，停止を定義します．

書式　1) KEY〔(〈キー番号〉)〕　ON

2) KEY〔(〈キー番号〉)〕　OFF

3) KEY〔(〈キー番号〉)〕　STOP

文例　KEY　ON

KEY(3)　ON

〈キー番号〉には，1から10までのファンクションキーの番号を指定します．(〈キー番号〉)を省略(カッコも含む)した場合には，すべての,ファンクションキーを指定したことになります．

書式1）割り込みを許可します．以後指定されたファンクションキーを押すごとに割り込みが発生し，ON　KEY　GOSUBによって定義された処理ルーチンに分岐します．

書式2）割り込みを禁止します．以後指定されたファンクションキーを押しても処理ルーチンへの分岐は起こりません(ファンクションキーの通常の動作になります)．

書式3）割り込みを停止します．以後指定されたファンクションキーを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後　KEY(n)　ONによって割り込みが許可されると，前にファンクションキーを押したことが有効となり，処理ルーチンに分岐します．

注意：プログラムの終了時にはKEY　OFFを実行しておいてください．さもないと本来のファンクションキーの機能が失われます．

参照：ON　KEY　GOSUB，サンプルプログラム30

# KILL　C

機能　ディスク上のファイルを削除します．

書式　KILL　〈ファイルディスクリプタ〉

文例　KILL　"B：DATA1"

KILL　"¥BASICC¥DATE1"

〈ファイルディスクリプタ〉で指定したファイルをディスクから削除します．

削除するファイルはクローズ状態になっていなければなりません．また，1つのKILLでは1つのファイルしか削除できません．すなわち，〈ファイルディスクリプタ〉中にワイルドカードを指定することはできません．

KILL はすべてのディスクファイルについて使用できますが，〈ファイルディスクリプタ〉には拡張子(もしあれば)まで含めたフルネームを指定する必要があります．

**注意**：オープン状態のファイルを削除しようとすると "File already open" エラーとなります．

参照：FILES／LFILES

# KINPUT

C

| | |
|---|---|
| 機　能 | キーボードから入力された２バイト系日本語文字を，文字変数に代入します． |
| 書　式 | KINPUT 〈変数名〉 |
| 文　例 | KINPUT　A$ |

〈変数名〉には，文字変数を指定します．

KINPUT が実行されると，自動的に日本語入力モードに入り，キーボードからの入力待ちの状態になります．日本語の入力方法の詳細に関しては，MS-DOS のマニュアルを参照してください(BASIC における日本語入力の操作法は MS-DOS のそれと互換性があります)．

**注意**：KINPUT は，他の INPUT 関連の命令と異なり，１つの文では１つの変数しか指定できません．また，日本語文字以外の文字を入力することもできません．

KINPUT での入力は偶数桁からになりますので，LOCATE で奇数桁を指定した場合は，その桁数に１を加えた桁が指定されたものと解釈されます．

参照：INPUT

# KINSTR 関数

C

| | |
|---|---|
| 機　能 | ２バイト系日本語文字を含む文字列の中から指定文字列を捜して，その文字の位置を得ます． |
| 書　式 | KINSTR(〔〈位置〉,〕〈文字列 1〉,〈文字列 2〉) |
| 文　例 | PRINT　KINSTR(3, A$,"番号")<br>NUM＝KINSTR("東京名古屋京都大阪", STA$) |

〈文字列 1〉の中から〈文字列 2〉が捜され，見つかればその位置が，見つからなければ 0 が，それぞれ値として得られます．

〈位置〉には，〈文字列 1〉中で捜し始める位置を，文字数(整数値)で指定します．このとき，

2バイト系日本語も1文字として数えます．〈位置〉が省略された場合は1が指定されたものとみなされ，〈文字列1〉の最初から捜し始めます．

〈文字列2〉に""(ヌルストリング)を指定すると，KINSTRの値は〈位置〉と同じになります．

注意：この関数は日本語キャラクタモードでのみ使用できます．グラフィックキャラクタモードで使用された場合〝Illegal function call″エラーになります．

参照：INSTR

## KLEN 関数 Ⓒ

**機能** 2バイト系日本語文字を含む文字列中の，特定タイプの文字の合計数を得ます．

**書式** KLEN(〈文字列〉〔,〈機能〉〕)

**文例**
```
PRINT  KLEN(A$, 1)
LN=KLEN("東京 STATION")
```

〈機能〉に指定する値(整数値)により，特定のタイプの文字の合計数を得ます．

0 ： 全体の文字数(2バイト系日本語文字も1文字として数える)
1 ： 1バイト系英数カナ文字の文字数
2 ： 2バイト系日本語文字の文字数
3 ： 2バイト系全角文字の文字数
4 ： 2バイト系半角文字の文字数

〈機能〉を省略した場合は0と解釈されます．

なお，2バイト系日本語文字のうち，全角の文字を2バイト系全角文字といい，半角のものを2バイト系半角文字といいます．

注意：この関数は日本語キャラクタモードでのみ使用できます．グラフィックキャラクタモードで使用された場合，〝Illegal function call″エラーになります．

参照：KTYPE，サンプルプログラム38

## KMID$ 関数 C

**機能** 2バイト系日本語文字を含む文字列の中から，任意の長さの文字列を抜き出します．

**書式** KMID$(〈文字列〉，〈式1〉〔，〈式2〉〕)

**文例** K$=KMID$(A$, 4, 3)

PRINT KMID$("JAPAN日本USA米国", 1, 5)

〈文字列〉中の，〈式1〉番目の文字から始まる〈式2〉文字の長さの文字列を得ます．ここで，〈式1〉，〈式2〉に指定する文字数の単位は，1バイト系英数カナ文字ならば1バイトを1文字として，2バイト系日本語文字ならば2バイトを1文字として，それぞれ数えます．

〈文字列〉の文字数が〈式1〉より小さい場合，KMID$の値はヌルストリング(空の文字列)となります．

〈式2〉を省略した場合や，〈文字列〉中の〈式1〉番目の文字から右の文字数が〈式2〉より小さい場合は，〈文字列〉中の〈式1〉番目の文字から始まる右のすべての文字列が結果として得られます．

**注意**：この関数は日本語キャラクタモードでのみ使用できます．グラフィックキャラクタモードで使用された場合，"Illegal function call" エラーになります．

**参照**：KLEN，MID$，MID$(関数)，サンプルプログラム35，36

## KNJ$ 関数 C

**機能** 漢字コード文字列4桁を2バイト系日本語文字1文字に変換します．

**書式** KNJ$(〈文字列〉)

**文例** PRINT KNJ$("3441")

〈文字列〉に指定した漢字コード4桁を，2バイト系日本語文字1文字に変換します．このとき，〈文字列〉の内容は次の条件をすべて満たしていなくてはなりません．1つでも満たされていないと，"Illegal function call" エラーになります．

(1) 文字列が4桁以上であること

(2) 最初の4桁の文字が，いずれも16進数(0～F)の範囲内の文字であること

(3) 4桁の文字の組み合せが，漢字コードの範囲内であること

〈文字列〉が4桁を超える場合，最初の4桁のみが用いられます．

注意：この関数は日本語キャラクタモードでのみ使用できます．グラフィックキャラクタモードで使用された場合，“Illegal function call” エラーになります．

参照：JIS$，サンプルプログラム34，36

# KPLOAD

C

| 機 能 | 利用者定義文字パターンをシステムに登録します． |
|---|---|
| 書 式 | KPLOAD <漢字コード>，<整数型配列名> |
| 文 例 | KPLOAD &H762A，CHRPTN% |

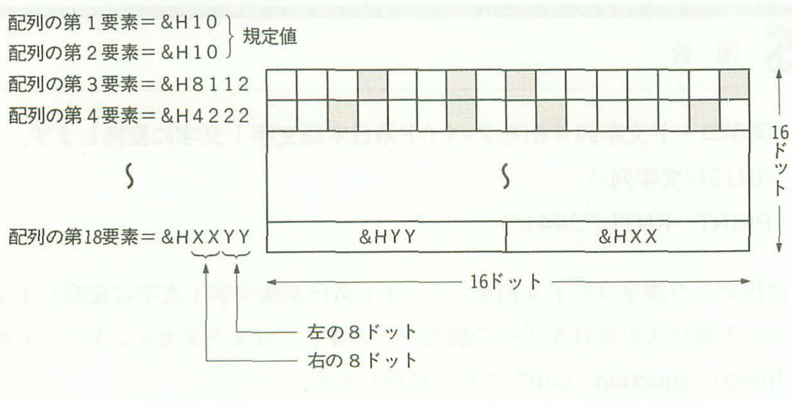
利用者が作字した利用者定義文字のパターンをシステムに登録する命令です．

<漢字コード>には登録するコードを指定します．指定可能な値の範囲は，&H7621～&H767E および&H7721～&H777E までです．この範囲にないときは，“Illegal function call”エラーとなります．

<整数型配列名> には文字パターンの格納された配列変数を指定します．<整数型配列名> に指定する配列変数は，あらかじめ DIM により，1 次元 18 要素の整数型配列として宣言しておいてください．

配列内に格納されている文字パターンは次のような形式でなければなりません．

配列の第 3 要素から第 18 要素までに，次のように実際の文字パターンのドットイメージを格納してください．




注意：<漢字コード>の範囲は，機種により&H7621～&H765F の範囲に制約されるものがありますが，BASIC は&H7621～&H767E，&H7721～&H777E の範囲しかチェックしませんので注意してください．

参照：DIM，サンプルプログラム34

## KTYPE 関数 C

機能　2バイト系日本語文字を含む文字列中の，指定位置の文字のタイプを得ます．

書式　KTYPE(〈文字列〉，〈式〉)

文例　B=KTYPE(A$，5)
　　　PRINT　KTYPE("日本語"，2)

〈文字列〉中の〈式〉バイト目の文字のタイプを得ます．

〈式〉には，タイプを得たい文字の位置を，〈文字列〉の先頭から数えたバイト数で指定します．この場合，1バイト系英数カナ文字は1文字を1バイトと数えますが，その他の文字はすべて1文字を2バイトと数えます．

得られる値とタイプとの関係は次のとおりです．

| 得られる値 | | タイプ |
|---|---|---|
| 0 | ： | 1バイト系英数カナ文字 |
| 1 | ： | 2バイト系全角文字 |
| 2 | ： | 2バイト系半角文字 |

〈式〉の値が0または〈文字列〉の文字数より大きい場合，あるいは〈文字列〉がヌルストリング(空の文字列)の場合は，"Illegal　function　call"エラーとなります．

注意：この関数は日本語キャラクタモードでのみ使用できます．グラフィックキャラクタモードで使用された場合，"Illegal function call"エラーになります．

参照：KLEN，サンプルプログラム38

## LEFT$ 関数 C

機能　文字列の左側から任意の長さの文字列を抜き出します．

書式　LEFT$(〈文字列〉，〈式〉)

文例　B$=LEFT$(A$，4)
　　　PRINT　LEFT$("standing"，5)

〈文字列〉の左側(最初)から〈式〉で指定した桁数の文字列を抜き出します．〈式〉の値は0から255の範囲になければなりません．

〈式〉の値が0ならば，LEFT$の値はヌルストリング(空の文字列)となります．〈式〉が〈文字列〉の文字数より大きければ，LEFT$の値は〈文字列〉とまったく同じになります．

参照：MID$，RIGHT$，サンプルプログラム25

## LEN 関数 C

機能　文字列の合計バイト数を得ます.

書式　LEN(〈文字列〉)

文例　PRINT　LEN(A$)

L=LEN("TEST")

〈文字列〉全体の合計バイト数を得ます.

出力されない文字や空白も数えられます. ここで, 出力されない文字とは, キャラクタコードで0から31までの, 通常コントロールコードと呼ばれるものをいいます.

LENでは, 〈文字列〉中にどんなタイプ(1バイト系, 2バイト系)の文字が含まれていても, 合計のバイト数が得られます.

参照 : KLEN, サンプルプログラム25

## LET C

機能　変数に値を代入します.

書式　〔LET〕〈変数名〉=〈式〉

文例　LET　I=I+1

I=(I+1)＊J

A$=STRING$(20,"＊")

LETはなくてもかまいません.

〈式〉の内容は, 数値, 文字列のいかなる型であってもかまいません. ただし, 〈変数名〉と〈式〉の型は同じでなければなりません. 型の異なる数値変数への代入では左辺の精度に合わせた代入が行われます.

注意 : 代入では必ず, 左辺に変数を置かなければなりません.

# LINE

**C**

**機　能**　指定した2点間に直線を描きます(パラメータ指定により四角形も描きます).

**書　式**　LINE 〔 | (Wx1, Wy1) / STEP(x1, y1) | 〕－ | (Wx2, Wy2) / STEP(x2, y2) | 〔,〈パレット番号1〉〕,〔, | B / BF | 〕〔, | 〈ラインスタイル〉 / 〈パレット番号2〉 / 〈タイルストリング〉 | 〕

**文　例**　LINE(100, 100)－(135, 100), 7,, &HF99F

LINE－STEP(30, 30), 7, BF, 7

ワールド座標系の2点(Wx1, Wy1)と(Wx2, Wy2)を結ぶ直線を描きます.

(Wx1, Wy1)が省略された場合, LP(最終参照点)の値を採用します. STEPを使って相対座標で指定することもできます.

〈パレット番号1〉には, 描く線の色を指定します. 省略された場合は, そのときCOLORで設定されているグラフィック画面のフォアグラウンドカラーが採用されます.

次のパラメータには, BかBFのどちらかを指定することができます. BはBoxの意味で, (Wx1, Wy1)と(Wx2, Wy2)の2点を対角とする四角形の箱を描きます. BFはBox Fillの意味で, 描いた四角形の内部をぬりつぶします.

〈ラインスタイル〉は, 描く線の形態を設定するパラメータであり, &H0から&HFFFF(16進数)の値をとることができます. この値とラインの形態との関係は, 図のとおりです. 〈ラインスタイル〉が省略された場合は, 直線を引きます. BFの指定時には指定できません.

〈パレット番号2〉は, 四角形の内部をぬりつぶす際の色を, 〈タイルストリング〉は, ぬりつぶす際の模様を指定します. 〈パレット番号2〉および〈タイルストリング〉は, BF指定のときだけ有効です. BF指定があり, 〈パレット番号〉, 〈タイルストリング〉のいずれも省略された場合には, 四角形を描いたのと同じ色で内部をぬりつぶします.

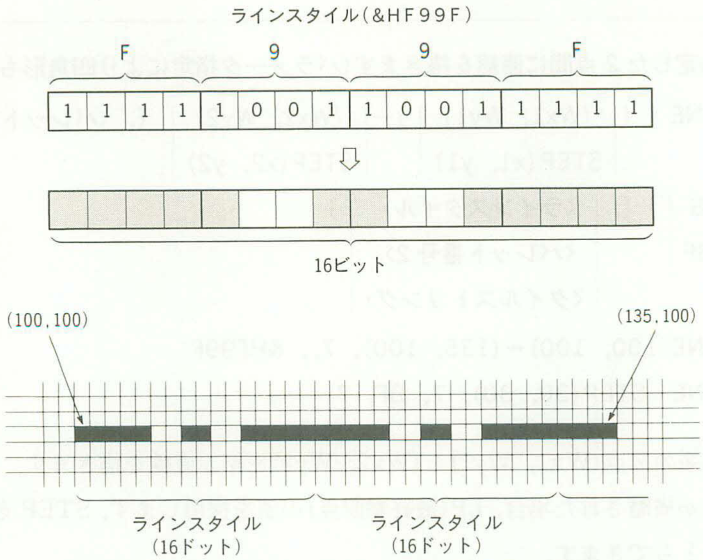
パレット番号については〔2〕COLORを, 〈タイルストリング〉については〔2〕PAINTを参照してください.

LINEを実行した場合, LPは(Wx2, Wy2)に移動します.

この図は文例を実行したものです. 〈ラインスタイル〉の16進数の値を2進数16ビットで表したものと画面上の16ドットが対応しています. "1" のビットに対応するドットが表示され, "0" のビットに対応するドットは表示されません. この例では, 水平座標の100から135までの36ドットの間に, 1点鎖線のラインスタイルが繰り返されています. 線を16ドット以上引いた場合には, ラインスタイルが画面上の16ドットごとに繰り返されることになります.

このようにラインスタイルを指定すると, 16ドット単位で線の形態を決めることができます

から，折れ線グラフや図面の作成の際，点線や１点鎖線などを簡単に描くことができます．




LINE(100,100)－(135,100),7,,&HF99F を実行した時

参照：〔2〕COLOR，〔2〕PAINT，サンプルプログラム15，16，17，18，20

# LINE INPUT　C

| | |
|---|---|
| 機　能 | キーボードから入力されるデータ(255バイト以内)を，区切ることなく一括して文字変数に代入します． |
| 書　式 | LINE　INPUT　〔<プロンプト文>；〕<文字変数名> |
| 文　例 | LINE　INPUT　"NAME"；NA$<br>LINE　INPUT　B$ |

LINE　INPUTでは，キーボードから入力されたすべてのデータがコンマなどの区切り記号も含めて<文字変数名>で指定された変数に代入されます(INPUTの場合は区切り記号で区切られたデータの１つ１つが順次抽出され，変数に代入されます)．

<プロンプト文>には，入力を受ける前に表示するメッセージを指定します．

入力を中断したい場合，STOPキーあるいはCTRL＋Cキーを入力します．プログラムの実行中に入力を中断すると，STOP　ONの状態にない限り，プログラムの実行も中断され，コマンドモードにもどります．このとき，それまでに入力した値はすべて無効となります(文字変数には値は代入されません)．

参照：INPUT，LINE　INPUT#，STOP　ON／OFF／STOP

## LINE INPUT#　C

機　能　シーケンシャルディスクファイルより，1行(255バイト以内)単位のデータを一括して文字型変数に読み込みます．

書　式　LINE　INPUT　#〈ファイル番号〉，〈文字型変数名〉

文　例　LINE　INPUT　#1，A$

〈ファイル番号〉には，OPENによって，そのファイルをオープンしたときに指定した番号を使います．

〈文字変数名〉は，データが代入される文字変数の名前です．

LINE　INPUT#は，シーケンシャルファイルの中から，キャリッジリターン(CR)コード＋ラインフィード(LF)コードで区切られている1行単位のデータを読み込みます．ここで，CRコードはCHR$(13)，LFコードはCHR$(10)に相当します．

LINE　INPUT#は，データファイルのそれぞれの行がスペースやコンマによっていくつかの欄に分割されているときや，アスキーセーブしたプログラムを他のプログラムのデータとして読み込むときなどに特に有効です．

参照：INPUT#，OPEN

## LINE INPUT@　G C

機　能　指定したトーカから送られてくる文字列データを受信して，文字変数に代入します．

書　式　1）LINE　INPUT@〈トーカアドレス〉〔，〈リスナアドレス〉〕；〈文字変数〉

2）LINE　INPUT@；〈文字変数〉

文　例　LINE　INPUT@1，3；A$

LINE　INPUT@；B$

書式 1）マスタモードの場合に使用します．

ATN（アテンション）をtrueにしてUNL（アンリスン）コマンド，トーカアドレス，リスナアドレス，MLA（マイリスンアドレス）を順次送出します．その後，ATNをfalseにして，指定されたトーカより送信されてくる文字列を受信し，デリミタ$C_R$で区切られるまでを一括して，文字変数に代入します．

書式 2）スレーブモードの場合に使用します．

スレーブモードでは，アドレスの指定ができないため，アドレスは省略しなければな

りません．この命令が実行されると，リスナがコントローラによりアドレスされるまで待ちます．MLA受信後，ATNがfalseになり，トーカから文字列が送出されてくると，そのデータを受信し，文字変数に代入します．

参照：INPUT@

# LINE INPUT WAIT　C

機　能　キーボードから入力されるデータを変数に代入します．その際，入力待ち時間を制限することができます．

書　式　LINE　INPUT　WAIT　<待ち時間>，〔<プロンプト文> {；|，} 〕<文字型変数名>

文　例　LINE　INPUT　WAIT　50，"No.="，NO$

LINE　INPUT　WAITは時間制限つきのLINE　INPUTです．したがって，LINE INPUTとINPUT　WAITの両方の機能を兼ねそなえています．詳しくはそれぞれの命令を参照してください．

参照：LINE　INPUT，INPUT　WAIT

# LIST／LLIST

機　能　メモリ上にあるプログラムの全部，または一部を表示あるいは印刷します．

書　式　1）LIST　〔<始点行番号>〕〔－<終点行番号>〕

2）LLIST　〔<始点行番号>〕〔－<終点行番号>〕

文　例　LIST　.

LLIST　100－200

LISTを実行すると，プログラムの内容が，書式1)は画面に，書式2)はプリンタに，それぞれ出力されます．

パラメータの指定方法とリストの関係は次のとおりです．

<始点行番号>－<終点行番号>　：　その範囲のすべてのプログラム行をリストする．

<始点行番号>－　：　始点行番号に始まりそれよりも大きい行番号のプログラム行すべてをリストする．

| | | |
|---|---|---|
| ー〈終点行番号〉 | ： | 最も若い行番号のプログラム行から終点行番号までをリストする． |
| 〈始点行番号〉 | ： | その行番号のプログラム行だけをリストする． |
| 行番号を省略 | ： | プログラム行をすべてリストする． |

行番号にピリオド(．)を指定すると，BASICインタプリタ内のポインタが示している現在の行を指します(たとえば，最後に修正された行)．

リストの出力は STOP キーまたは CTRL ＋ C の入力で終ります．

この命令は，実行し終るといつでもコマンドレベルにもどります．

コンパイラにおいてはこの機能は使用できません．

# LOAD

**機 能**　プログラムをメモリにロードします．

**書 式**　LOAD 〈ファイルディスクリプタ〉〔, R〕

**文 例**　LOAD "B : DEMO", R
　　　　LOAD "COM : N82NN"

〈ファイルディスクリプタ〉には，メモリにロードするプログラムファイル(ディスク，RS-232C回線，キーボード)を指定します．LOADを実行すると，すべての開いているファイルは閉じられ，変数に代入されている値は失われます．

Rオプションをつけると，ファイルは開いたままで，プログラムをロード後，直ちに実行を開始します．

なお，〈ファイルディスクリプタ〉に，拡張子".BAS"のついているファイルを指定する場合，その拡張子を省略することができます．

コンパイラにおいてはこの機能は使用できません．

注意：RS-232C回線からのプログラムロードの場合，プログラムのロードが完了しても，自動的にLOADコマンドは終了しません． STOP キーで強制的にブレーク(中止)しなければなりません．

参照：BLOAD，BSAVE，RUN，SAVE

## LOC 関数 Ⓒ

| | |
|---|---|
| 機能 | ファイル中での論理的な現在位置を得ます. |
| 書式 | LOC(〈ファイル番号〉) |
| 文例 | LAST=LOC(2) |

〈ファイル番号〉で指定されたファイルの種類によって，得られる値の意味が異なります.

**ランダムディスクファイルの場合**

最後に読み書き(GET／PUT)を行ったレコードのレコード番号が得られます.

**シーケンシャルディスクファイルの場合**

オープンしてから現在までに読み書きしたレコード数が得られます.

**キーボードファイルの場合**

割り込みによって受け付けられ，キーボードバッファ中にたまっている文字数が得られます.

**RS-232C 回線ファイルの場合**

割り込みによって受け付けられ，入力バッファ中にたまっている文字数が得られます.

読み取り動作(INPUT #，INPUT$など)が行われずに，割り込みによる RS-232C 回線からの受信が続くと，やがてバッファが一杯になりエラーとなるため，これを監視するのに LOC を使用することができます.

参照：EOF，LOF

## LOCATE Ⓒ

| | |
|---|---|
| 機能 | テキスト画面のカーソルを指定位置へ移動します. |
| 書式 | LOCATE 〔〈X〉〕〔，〈Y〉〕〔，〈カーソルスイッチ〉〕 |
| 文例 | LOCATE 10, 10<br>LOCATE ,, 0 |

X は水平座標を表し，省略された場合は 0 とみなされます. Y は垂直座標を表し，省略された場合は現在カーソルがセットされている行とみなされます. これらは，ともに画面の左上を(0，0)とするキャラクタ座標により指定します.

〈カーソルスイッチ〉はカーソルの表示状態を決めるスイッチで，0 を指定すると表示されなくなり，1 を指定すると表示されるようになります.

参照：サンプルプログラム10，33

## LOF 関数

C

| | |
|---|---|
| 機能 | ファイルの大きさを得ます. |
| 書式 | LOF(〈ファイル番号〉) |
| 文例 | MAXREC=LOF(2) |

〈ファイル番号〉で指定されたファイルの種類によって，得られる値の意味が異なります.

**シーケンシャルディスクファイルの場合**

ファイルの大きさがバイト数で得られます.

**ランダムディスクファイルの場合**

ファイルの最大レコード番号が得られます.

**RS-232C 回線ファイルの場合**

入力バッファの残りのバイト数(入力可能な余裕バイト数)が得られます.

参照：EOF，LOC，サンプルプログラム29

## LOG 関数

C

| | |
|---|---|
| 機能 | 自然対数を得ます. |
| 書式 | LOG(〈数式〉) |
| 文例 | PRINT　LOG(35／9)<br>LA=LOG(X＊Y) |

〈数式〉の，自然対数(e を底とした対数)を値として得ます.

〈数式〉に倍精度実数が含まれる場合，得られる値は倍精度となりますが，他の場合には単精度となります.

参照：EXP

## LPOS 関数 C

機能　現在のプリンタのヘッド位置を得ます.

書式　LPOS(〈式〉)

文例　HEAD=LPOS(0)

使用中のプリンタの、現在のヘッド位置(桁位置，すなわち水平位置)を得ます.

〈式〉は，ダミーであって意味を持ちませんが，省略できません.

参照：WIDTH　LPRINT

## LPRINT C

機能　プリンタにデータを出力します.

書式　LPRINT　〔〈式〉〕〔 { , | ; } 〈式〉…〕〔 { , | ; } 〕

文例　LPRINT　A$, D2

LPRINT　CHR$(&H41); CHR$(&H42); CHR$(&H43)

指定した式の値や文字列などのデータを，プリンタに出力します.

LPRINTは，データをプリンタに出力する点が違うだけで，その動作はPRINTと同じです.

参照：PRINT

## LPRINT USING C

機能　文字列，数値などのデータを編集し，プリンタに出力します.

書式　LPRINT　USING　〈書式制御文字列〉;〈式〉〔 { , | ; } 〈式〉…〕〔 { , | ; } 〕

文例　LPRINT　USING　"@　###"; A$, B

LPRINT　USING　"¥¥#.###"; C

〈書式制御文字列〉によって決定される領域や書式に従って，〈式〉に指定された各種データをプリンタに出力します.

LPRINT　USINGは，データをプリンタに出力する点が違うだけで，その動作はPRINT

USINGと同じです.

参照：PRINT　USING

## LSET／RSET

Ⓒ

| 機　能 | ランダムファイルバッファのフィールドにデータを代入します. |
|---|---|
| 書　式 | 1) LSET　〈文字変数〉=〈文字列〉<br>2) RSET　〈文字変数〉=〈文字列〉 |
| 文　例 | LEFT　A$="単価"<br>RSET　B$=MKS$(I) |

〈文字変数〉には, FIELDでランダムファイルバッファ上に割り当てられたフィールド変数を指定します.

〈文字列〉を指定することにより，フィールド変数に対応するフィールドにデータ(文字型)を代入することができます.〈文字列〉の文字数がFIELDで割り当てられたフィールド変数の長さより短い場合, LSETでは左詰め, RSETでは右詰めでフィールドを満たします. また,〈文字列〉がフィールドより長い場合は, LSET, RSETとも右側の部分が失われます.

注意：〈文字変数〉は，あらかじめFIELDで定義されていなければなりません. 定義されていない変数名を用いると，データを正しく代入することができません.

また，〈文字列〉に指定するデータは文字型でなければなりませんので，数値型データはMKI$, MKS$, MKD$のいずれかにより文字型データに変換しておかなければなりません.

参照：FIELD, GET, MKI$／MKS$／MKD$, PUT, サンプルプログラム26, 29

## MAP　関数

Ⓒ

| 機　能 | スクリーン座標，ワールド座標の相互変換を行います. |
|---|---|
| 書　式 | MAP(〈数式〉,〈機能〉) |
| 文　例 | SY1=MAP(WY1, 1) |

〈数式〉で指定されるスクリーン座標あるいはワールド座標の値を,〈機能〉の指定に従い，対応するワールド座標あるいはスクリーン座標に変換します.

〈機能〉に指定する値により，次のような変換を行います.

0 ： 〈数式〉をワールド座標系における x 座標とみなし，これを対応するスクリーン座標系での x 座標に変換する(Wx → Sx).

1 ： 〈数式〉をワールド座標系における y 座標とみなし，これを対応するスクリーン座標系での y 座標に変換する(Wy → Sy).

2 ： 〈数式〉をスクリーン座標系における x 座標とみなし，これを対応するワールド座標系での x 座標に変換する(Sx → Wx).

3 ： 〈数式〉をスクリーン座標系における y 座標とみなし，これを対応するワールド座標系での y 座標に変換する(Sy → Wy).

MAP を用いれば，ワールド座標からスクリーン座標への変換ができますから，GET や PUT のようにスクリーン座標で位置を指定する操作の場合にも，ワールド座標を変換して指定することができます.

注意：MAP は，変換の結果がスクリーン座標系あるいはワールド座標系から外れてもエラーとなりません.

参照：VIEW，VIEW(関数)，WINDOW，WINDOW(関数)

# MERGE

| | |
|---|---|
| 機能 | メモリ上のプログラムに，ディスク上のプログラムファイルを混合します. |
| 書式 | MERGE 〈ファイルディスクリプタ〉 |
| 文例 | MERGE "B : TEST" |

メモリ上のプログラムと，〈ファイルディスクリプタ〉により指定したディスク上のプログラムファイルを混合(マージ)して１つのプログラムにし，メモリ上に置きます.

メモリ上のプログラムと，ディスク上のプログラムファイルに同一の行番号があった場合には，ディスク上のプログラムファイル中の行が採用されます.

なお，マージ後，ただちにそのプログラムを実行させたいときは，MERGE オプションつきの CHAIN を使うことができます.

この機能はコンパイラにおいては使用できません.

注意：混合するプログラムファイルはアスキーセーブされていなければなりません．そうでない場合には，"Sequential I/O only" エラーになります.

参照：CHAIN，LOAD，SAVE

# MID$ C

機能　文字列の一部を置き換えます．

書式　MID$(〈文字変数〉，〈式 1〉〔，〈式 2〉〕)＝〈文字列〉

文例　MID$(A$，3)＝"ABC"

MID$(B$，N，2)＝C$

〈文字変数〉に代入されている文字列の〈式 1〉番目の文字から〈式 2〉個の文字を，〈文字列〉の最初から〈式 2〉個分の文字列で置き換えます．

〈式 1〉は 1 から 255 の範囲，また，〈式 2〉は 0 から 255 の範囲になければなりません．

〈文字列〉には文字変数を用いることもできます．

〈式 2〉が省略された場合，または〈式 2〉の値が〈文字列〉の文字数を超える場合，〈文字変数〉に代入されている文字列を，〈文字列〉のすべての文字と置き換えます．

**注意：** この命令を，1 バイト系と 2 バイト系の混在文字列に使用する場合は，文字の境界に充分注意しないと，予期しない結果となります．

**参照：** MID$(関数)

# MID$ 関数 C

機能　文字列の中から任意の長さの文字列を抜き出します．

書式　MID$(〈文字列〉，〈式 1〉〔，〈式 2〉〕)

文例　B$＝MID$(A$，2，3)

COUNTRY$＝MID$("JPNUSAFRAGBRCAN"，N＊3＋1，3)

〈文字列〉の〈式 1〉番目の文字から〈式 2〉桁の文字列を抜き出します．

〈式 1〉は 1 から 255 の範囲，また，〈式 2〉は 0 から 255 の範囲になければなりません．

〈式 2〉を省略した場合や，〈文字列〉の〈式 1〉番目の文字から右側全部の文字数が〈式 2〉より小さい場合は，〈文字列〉の〈式 1〉番目の文字より右側全部の文字列を抜き出します．

〈式 1〉が〈文字列〉の文字数より大きければ，MID$の値はヌルストリング(空の文字列)となります．

**注意：** この命令を，1 バイト系と 2 バイト系の混在文字列に使用する場合は，文字の境界に充分注意しないと，予期しない結果となります．

**参照：** LEFT$，MID$，RIGHT$，サンプルプログラム23，25

# MKDIR　C

| | |
|---|---|
| 機　能 | ディスクに新しいディレクトリを作成します． |
| 書　式 | MKDIR 〔<ドライブ名>〕<ディレクトリ名> |
| 文　例 | MKDIR　"BIN"<br>MKDIR　"B : ¥SOURCE"<br>MKDIR　"A : BIN¥SUB"<br>MKDIR　"..¥TEST" |

<ドライブ名>で指定されたドライブにあるディスクに，<ディレクトリ名>で指定される新しいディレクトリを作成します．<ドライブ名>が省略された場合は，カレントドライブにあるディスクが指定されたものとみなされます．

<ディレクトリ名>には，作成したいディレクトリの名前を，¥（ルート）で始まる絶対指定か，現在のカレントディレクトリからの相対指定によって指定します．

例）　MKDIR　"B : ¥SOURCE"

Bドライブにあるディスクのルートディレクトリ内にSOURCEという名のディレクトリを作成（絶対指定）します．

MKDIR　"BIN"

カレントドライブにあるディスクのカレントディレクトリ内にBINという名のディレクトリを作成（相対指定）します．

ディレクトリが階層化されている場合には，複数の<ディレクトリ名>を¥でつないで指定します．ただしこの場合，作成しようとするディレクトリ名（いちばん最後に指定するディレクトリ名）以外は，すでに存在していなくてはなりません．

例）　MKDIR　"A : ¥BIN¥SUB"

Aドライブにあるディスクのルートディレクトリのさらに下にあるBINディレクトリ内にSUBという名のディレクトリを作成します．ただしこの場合，BINディレクトリはあらかじめ存在していなくてはなりません．

なお，カレントディレクトリのすぐ上にあるディレクトリを，".."（ピリオド2個）で表すことができますので，相対指定を行う場合に利用することができます．

例）　MKDIR　"..¥TEST"

カレントドライブにあるディスクのカレントディレクトリのすぐ上にあるディレクトリ内にTESTという名のディレクトリを作成します．

参照：CHDIR，RMDIR

# MKI$／MKS$／MKD$ 関数 Ⓒ

**機能** 数値データをその数値の内部表現に対応した文字列に変換します．

**書式** 1）MKI$(〈整数値〉)

2）MKS$(〈単精度実数値〉)

3）MKD$(〈倍精度実数値〉)

**文例**

```
A$=MKI$(16961)
LSET  B$=MKS$(1.23)
RSET  C$=MKD$(3.141592654#)
```

これらの関数は，数値データをランダムディスクファイルに対して書き出す際，数値データはそのままの形では書き出すことができないため，これを文字型データに変換するために使用するものです．

MKI$ ： 整数値を2文字(2バイト)の文字列に変換(CVIの逆)．

MKS$ ： 単精度実数値を4文字(4バイト)の文字列に変換(CVSの逆)．

MKD$ ： 倍精度実数値を8文字(8バイト)の文字列に変換(CVDの逆)．

実際にランダムディスクファイルに数値データを書き出す際には，まずこれらの関数を用いて数値データを文字型化し，これをLSET／RSETでフィールド変数にセットしてから，PUTを行います．

これらの関数で文字列に変換された数値データを，もとの数値型に変換するには，CVI／CVS／CVDの各関数を使用します．

数値から文字への変換は数値の内部表現の値をそのままそれに対応する文字コードをもつ文字列にすることによって行われます．実際の数値と変換後の文字列との関係を整数値で説明すると次のようになります．

整数値が16961の場合，この数値の内部表現は&H4241(16進表記)であり，A$=CHR$(&H41)+CHR$(&H42)，すなわち，A$="AB"となります(上下バイトが逆になることに注意)．つまり，数値データをキャラクタコードとみなすわけです．

**注意**：STR$関数とは変換のしかたがまったく異なります．

**参照**：CVI／CVS／CVD，LSET／RSET，STR$，サンプルプログラム26

# MOUSE C

機　能　　マウスの種々の動作環境を設定します．

書　式　　1) MOUSE　0

2) MOUSE　1〔, Sx〕〔, Sy〕〔, 〈カーソルスイッチ〉〕

3) MOUSE　2, 〈X 指示点〉, 〈Y 指示点〉, 〈XOR 文字列〉

4) MOUSE　3, 〈方向〉, 〈移動比率〉

5) MOUSE　4, Sx1, Sy1, Sx2, Sy2

6) MOUSE　5, 〈色〉

7) MOUSE　6

文　例　　MOUSE　0

MOUSE　1, 100, 100, 1

MOUSE　2, 0, 0, A$

MOUSE　3, 0, 200

MOUSE　4, 50, 50, 150, 150

MOUSE　5, 0

MOUSE　6

マウスの種々の動作環境を設定するための命令です．次に，各書式の機能を説明します．

書式 1) MOUSE　0

マウスカーソルの形状や移動比率(マウスカーソルの移動の速さ)，移動範囲などについてマウスの初期化をします．

他の MOUSE 命令や MOUSE 関数を実行する前には，必ず MOUSE　0 を実行する必要があります．MOUSE　0 を実行せずに他の MOUSE 命令あるいは MOUSE 関数を実行すると，"MOUSE　not　initialized" エラーになります．

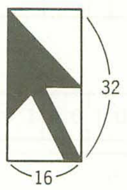
マウスカーソルは現在のアクティブページに設定されます．MOUSE　0 の実行後，マウスカーソルの表示位置は画面の中心になっています．ただし，MOUSE　0 では，マウスカーソルは表示されません．

MOUSE　0 実行時のマウスカーソルの初期状態は次のとおりです．

| | |
|---|---|
| 表示位置 | 画面の中央 |
| 移動比率 | 8 |
| 移動範囲 | グラフィック画面全体 |
| 色 | パレット番号 4 に対応した色 |

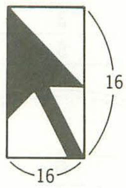
形状と指示点

600×400モードの時




指示点(0,0)

600×200モードの時




指示点(0,0)

書式 2）MOUSE　1

マウスカーソルの表示位置と表示／非表示を決めます．

Sx，Syを指定すると，マウスカーソルはスクリーン座標上の（Sx，Sy）によって表される位置に表示されます．SxまたはSyが省略された場合は，現在のマウスカーソルのX座標（水平座標）またはY座標（垂直座標）が設定されたのと同じになります．

〈カーソルスイッチ〉には，1を指定するとカーソルが表示され，0を指定すると表示されなくなります．省略した場合は，現在のカーソルの状態が引き継がれます．

例）MOUSE　1，100，100，1

マウスカーソルが（100，100）の位置に表示される．マウスを移動させるとマウスカーソルはこの位置から動きはじめる．

MOUSE　1，，0，1

現在のマウスカーソルのX座標と，Y座標0で示される位置にマウスカーソルが表示される．

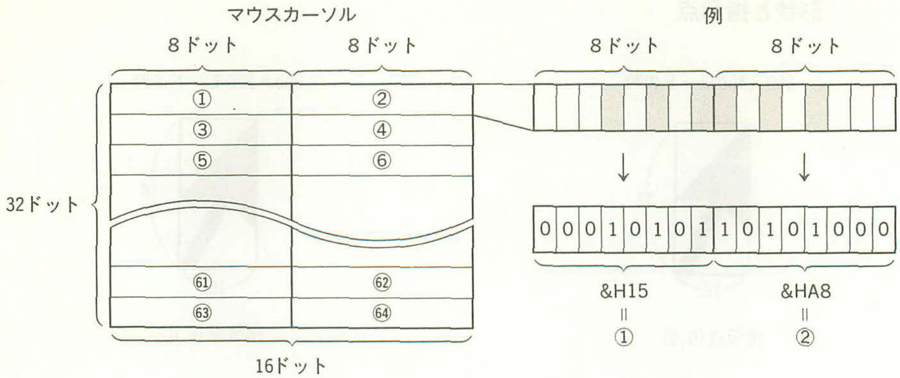
MOUSE　1，，，0

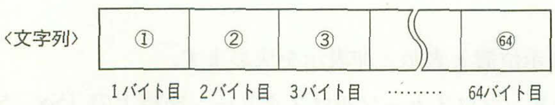
現在表示されているマウスカーソルの表示をとりやめる．

書式 3）MOUSE　2

マウスカーソルの形状を〈XOR文字列〉で，指示点を〈X指示点〉と〈Y指示点〉で指定します．

マウスカーソルの形状は16×32ドットの範囲内で任意に定義することができます．形状の定義は，マウスカーソルの1ドットのON／OFFを〈XOR文字列〉の1ビットのON／OFFに対応させて行います．




図のように，64 バイトの〈XOR 文字列〉で，マウスカーソルの形状を定義します．〈XOR 文字列〉の 1 バイト分でマウスカーソルの 8 ドット分を定義するわけです．すなわち，この図の場合における〈XOR 文字列〉の指定方法は次のようになります．

&H15＋&HA8＋……

↓

MOUSE　2，0，0，CHR$ (&H15)＋CHR$ (&HA8)＋……

マウスカーソルの指示点とは，マウスカーソルの表示位置 (MOUSE　1参照) に対応する点です．指示点はマウスカーソルの 16×32 ドットの範囲の任意の 1 点(1 ドット)を選択することができます．

〈X 指示点〉(水平方向) は 0～15，〈Y 指示点〉(垂直方向) は 0 から 31 の範囲で指定します．

**書式 4) MOUSE　3**

マウスカーソルの移動比率 (移動の速さ) を決めます．移動比率とはマウスカーソルの移動距離に対するマウスの移動距離の割合です．

〈移動比率〉にはマウスカーソルを 8 ドット移動させるのに必要なマウスの移動距離を，0～32767 の範囲の整数で指定します．標準値は 8 です．〈移動比率〉の値を大きくすると移動速度がおそくなり，小さくすると速くなります．たとえば 4 を指定すると，標準値の 2 倍の速度になり，逆に 16 を指定すると 1／2 の速度になります．〈移動比率〉に 0 を指定すると，移動比率は 8 にもどります．

〈方向〉は，移動比率を変える方向を示します．0 を指定すると X (水平) 方向の移動比率が，1 を指定すると Y (垂直) 方向の移動比率が変わります．

書式 5) MOUSE　4

マウスカーソルはグラフィック画面全体にわたって移動させることができますが，マウスの動きに関係なく，マウスカーソルの移動範囲を縮めたり広げたりすることができます．

移動範囲は（Sx1，Sy1）と（Sx2，Sy2）の2点を対角とする四角形で表します．

例）MOUSE　4，0，0，100，100

マウスカーソルの移動範囲が画面左上の（0，0）と（100，100）を対角とする四角形の内部になります．

MOUSE　4を実行すると，マウスカーソルが移動範囲の境界まで動いた後，マウスをいくら境界の方向に動かしてもマウスカーソルは境界のところで止まったままになります．

なお，移動範囲を設定した後に，MOUSE　1で移動範囲外の座標を指定すると，マウスカーソルは移動範囲内で，MOUSE　1で指定された座標に最も近い点に表示されます．

書式 6) MOUSE　5

マウスカーソルの色を指定します．

〈色〉でマウスカーソルの色を指定します．〈色〉は0～3の値で指定し，それぞれ次のようにパレットに対応しています．

0　：　パレット番号1
1　：　パレット番号2
2　：　パレット番号4
3　：　パレット番号8

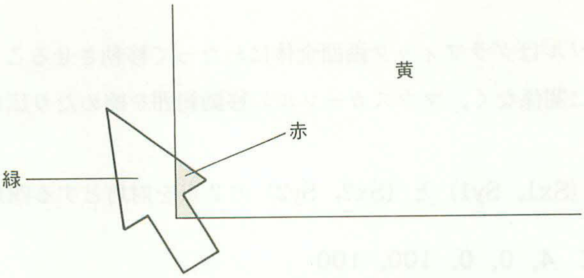
ただし，3の指定は4096色中•16色モードのときのみ有効です．8色中•8色モードあるいは4096色中•8色モードのときに3を指定するとエラーになります（カラーモードの詳細は〔1〕COLORを参照してください）．

パレットが初期状態のとき，0を指定すると青，1を指定すると赤，2を指定すると緑でマウスカーソルが表示されます．

例）MOUSE　5，0

マウスカーソルの色がパレット番号1に対応した色になります．

なお，マウスカーソルとグラフィク情報の間にはXORの関係が保たれています．このため，マウスカーソルがグラフィック情報と重なると，重なった部分の色が変化します．たとえば，次のように黄色で塗られたグラフィック情報と緑色カーソルが重なると，重なった部分の色は赤になります（パレットが初期状態の場合）．




これは次の関係によります.

| | G | R | B | | |
|---|---|---|---|---|---|
| | 1 | 1 | 0 | → | 黄 |
| XOR) | 1 | 0 | 0 | → | 緑 |
| | 0 | 1 | 0 | → | 赤 |

ただし, マウスカーソルが移動した後は, グラフィック情報は元の色にもどります.

**書式 7) MOUSE　6**

マウスカーソルの表示をとりやめ, マウス機能をOFFの状態にします.

マウス使用後には, 必ずこの命令を実行してください. MOUSE　6の実行後, マウスを使用する場合は, 再びMOUSE　0を実行してマウスの初期化を行ってください.

**注意**：マウスを利用してLINEやCIRCLEなどグラフィック命令を実行する場合は, 次のことに注意してください.

(1) グラフィック命令を実行する直前にMOUSE　1,,,0を実行してマウスカーソルを消してください. マウスカーソルが表示されたままでマウスカーソルにグラフィック情報が重なると, マウスカーソルが移動した後, マウスカーソルとグラフィック情報が重なっていたところの色が変わることがあります.

(2) グラフィック命令実行後, MOUSE　1,,,1を実行するとマウスカーソルが表示されます.

(3) 白黒モードのときにMOUSE　5を実行すると, MOUSE　5で指定された色に対応するページにマウスカーソルが書き込まれます. たとえば, 高分解能白黒モードのとき, MOUSE　5, 1を実行するとパレット番号2に対応した色は赤（この場合はパレットとカラーコードの関係が初期状態とする）となり, これに対応するページ2またはページ5（4096色中•16色モードのときはページ2またはページ6）にマウスカーソルが書き込まれます. これを画面に表示させるには, SCREENのディスプレイページの指定でページ2またはページ5（4096色中•16色モードのときはページ2またはページ6）を表示させる必要があります.

参照：〔1〕COLOR, 〔2〕COLOR, MOUSE(関数), ON MOUSE GOSUB, MOUSE ON／OFF／STOP, SCREEN

# MOUSE 関数 C

機能　マウスからの情報を得ます．

書式
1) MOUSE (0)
2) MOUSE (1)
3) MOUSE (2, <ボタン番号>)
4) MOUSE (3, <ボタン番号>)
5) MOUSE (4, <ボタン番号>)
6) MOUSE (5, <ボタン番号>)
7) MOUSE (6, <ボタン番号>)
8) MOUSE (7, <ボタン番号>)
9) MOUSE (8, <ボタン番号>)
10) MOUSE (9)
11) MOUSE (10)

文例
A=MOUSE (0)
B=MOUSE (7, 1)

マウスの状態についての情報（マウスがどのような状態にあるか）を得ます．

マウスからの情報には，現在の座標やボタンが押されたかどうかなどがあります．

<ボタン番号>で1を指定すると左のボタンの情報を，2を指定すると右ボタンの情報を得ます．また，得られる座標の値は，すべてスクリーン座標上のものです．

**書式 1) MOUSE (0)**

マウスカーソルの現在のX座標（水平座標）を得ます．

**書式 2) MOUSE (1)**

マウスカーソルの現在のY座標（垂直座標）を得ます．

**書式 3) MOUSE (2, <ボタン番号>)**

<ボタン番号>で指定されたボタンの状態を得ます．

ボタンが押されているときは1，離されているときは0が得られます．

**書式 4) MOUSE (3, <ボタン番号>)**

最後にこの関数が実行されてから現在までの間に，<ボタン番号>で指定されたボタン

が押された回数を得ます．

書式 5）MOUSE（4,〈ボタン番号〉）

〈ボタン番号〉で指定されたボタンが最後に押されたときのX座標を得ます．

書式 6）MOUSE（5,〈ボタン番号〉）

〈ボタン番号〉で指定されたボタンが最後に押されたときのY座標を得ます．

書式 7）MOUSE（6,〈ボタン番号〉）

最後にこの関数が実行されてから現在までの間に，〈ボタン番号〉で指定されたボタンが離された回数を得ます．

書式 8）MOUSE（7,〈ボタン番号〉）

〈ボタン番号〉で指定されたボタンが最後に離されたときのX座標を得ます．

書式 9）MOUSE（8,〈ボタン番号〉）

〈ボタン番号〉で指定されたボタンが最後に離されたときのY座標を得ます．

書式10）MOUSE（9）

最後にこの関数が実行されてから現在までに，マウスカーソルが移動したX方向の移動距離を得ます．右向きが正の方向で左向きが負の方向となります．

書式11）MOUSE（10）

最後にこの関数が実行されてから現在までに，マウスカーソルが移動したY方向の移動距離を得ます．下向きが正の方向で上向きが負の方向となります．

参照：MOUSE，MOUSE　ON／OFF／STOP，ON　MOUSE　GOSUB

# MOUSE ON／OFF／STOP　C

| 機　能 | マウスによる割り込みの許可，禁止，停止を制御します． |
|---|---|
| 書　式 | 1）MOUSE(〈事象番号〉)　ON<br>2）MOUSE(〈事象番号〉)　OFF<br>3）MOUSE(〈事象番号〉)　STOP |
| 文　例 | MOUSE(1)　ON |

マウスの種々の動作による割り込みを許可(ON)するか，禁止(OFF)するか，停止(STOP)するかを宣言します．

〈事象番号〉と事象の関係は次のとおりです．

1 : マウスが移動した

2 : 左側のボタンが押された

3 : 右側のボタンが押された

4 : 左側のボタンが離された

5 : 右側のボタンが離された

書式 1) 〈事象番号〉で指定した割り込みを，この命令の次の命令を実行した後，許可します．その後，指定した事象が発生するごとに割り込みが発生し，ON MOUSE GOSUB で定義されている処理ルーチンへ分岐します．

書式 2) 割り込みを禁止します．その後，指定した事象が発生しても処理ルーチンへの分岐は起こりません．

書式 3) 割り込みを停止します．その後，指定した事象が発生してもそのことを覚えているだけで処理ルーチンへの分岐は起こりません．しかし，その後，MOUSE ON により割り込みが許可されると，処理ルーチンへ分岐します．

参照：MOUSE，ON MOUSE GOSUB

# NAME

C

| | |
|---|---|
| 機能 | ディスクファイルの名前を変更します． |
| 書式 | NAME 〈旧ファイルディスクリプタ〉 AS 〈新ファイルディスクリプタ〉 |
| 文例 | NAME "SAMPLE" AS "SAMPLE1"<br>NAME "B : DATA. BAS" AS "B : DATA1" |

〈旧ファイルディスクリプタ〉で表されているディスク上のファイルの名前を〈新ファイルディスクリプタ〉に変更します．

ディスクファイルに対するファイルディスクリプタの指定形式は，次のとおりです．

〔ドライブ名：〕ファイル名

ドライブ名が省略された場合，カレントドライブ上のディスクが処理の対象になります．

拡張子を持つファイルの名前を変更する場合，〈旧ファイルディスクリプタ〉には必ず拡張子も指定しなければなりません．

〈旧ファイルディスクリプタ〉と〈新ファイルディスクリプタ〉に指定されるドライブ名は同じでなければなりません．

なお，ファイルの名前の変更を行うファイルはクローズされた状態でなければなりません．

# NEW

| 書 式 | メモリ上にあるプログラムを抹消し，すべての変数を初期化します． |
|---|---|
| 書 式 | NEW |
| 文 例 | NEW |

NEW はメモリ上のプログラムを抹消し，すべての変数を初期化(クリア)します．また，そのとき開かれているファイルもすべて閉じられます．

コマンドの実行が終ると，いつもコマンドレベルにもどります．

コンパイラにおいてはこの機能は使用できません．

参照：CLEAR，RUN

# OCT$ 関 数 C

| 機 能 | 10 進数を 8 進数に変換し，その文字列を得ます． |
|---|---|
| 書 式 | OCT$(〈数式〉) |
| 文 例 | PRINT　OCT$(320) |

〈数式〉の値を 8 進数に変換して，その文字列を得る関数です．〈数式〉の値は，−32768 から 32767(または 0 から 65535)までの範囲になければなりません．〈数式〉の値と得られる文字列との対応は次のとおりです．

| 〈数式〉の値 | | 得られる 8 進表記文字列 |
|---|---|---|
| 0〜32767 | : | 0〜 77777 |
| −32768〜−1 | : | 100000〜177777 |
| 32768〜65535 | : | 100000〜177777 |

参照：HEX$，STR$，VAL，サンプルプログラム24

# ON COM GOSUB　C

機　能　RS-232C回線からの割り込みが発生したとき，分岐する処理ルーチンの開始行を指定します．

書　式　ON　COM〔(<回線番号>)〕　GOSUB　<行番号>

文　例　ON　COM　GOSUB　1000

ON　COM(2)　GOSUB　*RECV

指定したRS-232C回線に入力割り込みがあったときに分岐する処理ルーチンの開始行番号を定義します．

<回線番号>に指定できる値と意味は次のとおりです．(<回線番号>)を省略(カッコも含む)した場合は，第1回線となります．

1　：　RS-232C第1回線
2　：　RS-232C第2回線
3　：　RS-232C第3回線

ただし，2，3は専用のインタフェースボードが必要です．

<行番号>には分岐させる開始行番号を指定します．

処理ルーチンからの復帰は，一般のサブルーチンと同じで，RETURNで行います．たんにRETURNとしたときは中断した所から再開し，後ろに行番号を指定したときはその行から再開します．

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないようにしてください．

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みが発生したとき，COM　ONの状態でなければなりません．
ON　COM　GOSUB　は，RS-232C回線をOPENした後実行されなければ有効となりません．

参照：COM　ON／OFF／STOP，OPEN，RETURN

# ON ERROR GOTO C

**機能** エラーが起こったときに分岐する処理ルーチンの開始行を定義します.

**書式** ON ERROR GOTO 〈行番号〉

**文例** ON ERROR GOTO 1000

〈行番号〉には分岐させる処理ルーチンの開始行番号を指定します.

この命令を実行しておくと，エラーが起こったときに，指定した開始行から始まるエラー処理ルーチンに処理が移ります．エラー発生時のプログラム行番号はERLに，エラーコードはERRにセットされますので，エラー処理ルーチン内では，これらを利用して独自のエラー処理を行うことができます.

処理ルーチンから復帰するには，一般のサブルーチンとは異なり，RESUMEを使います.

プログラムの実行終了後は，必ずON ERROR GOTO 0を実行して，エラー割り込みを無効にしておくようにしてください．さもないと，エラーが発生するたびに(そのプログラムがメモリ上にある限り，ダイレクトモードであっても)エラー処理ルーチンに処理が移ってしまいます.

ON ERROR GOTO 0の実行後は，エラーが生じると通常どおりエラーメッセージが表示され，プログラムの実行は停止します.

エラー処理ルーチン内にON ERROR GOTO 0がある場合は，エラーによる分岐(エラートラップ)の原因となったエラーのエラーメッセージを表示し，プログラムの実行を停止します.

**注意**：〈行番号〉に指定された行が存在しない場合，“Undefined line number”エラーが起こります.

エラー処理ルーチンの中ではエラー割り込みは起こりません．エラー処理ルーチンの実行中にエラーが起きた場合には，それに対するエラーメッセージが表示され，実行は停止します.

**参照**：ERL／ERR，ERROR，RESUME，サンプルプログラム22

## ON…GOSUB／ON…GOTO　C

機能　指定されたいずれかの行に分岐します．

書式　1) ON 〈式〉 GOSUB 〈行番号〉〔,〈行番号〉…〕

　　　2) ON 〈式〉 GOTO 〈行番号〉〔,〈行番号〉…〕

文例　ON A GOSUB 100, 200, 300, 400

　　　ON N GOTO ＊ENTRY1, ＊ENTRY2, ＊ENTRY3, ＊ENTRY 4

〈式〉の値に応じて，〝〈式〉の値〞番目の〈行番号〉に処理が移ります．たとえば，〈式〉の値が3であったなら，行番号の並びの3番目の行が分岐先となります．

ON…GOSUBの場合，〈行番号〉はサブルーチン(RETURNで終っている必要がある)の開始行でなくてはなりません．

注意：〈式〉の値が負の場合には，〝Illegal function call〞エラーが起こりますが，値が0あるいは並びの行番号より大きい場合には次の行に実行が移り，エラーは起こりません．

参照：GOSUB，GOTO／GO TO，サンプルプログラム29，30

## ON HELP GOSUB　C

機能　HELP キーによる割り込み処理ルーチンの開始行を定義します．

書式　ON HELP GOSUB 〈行番号〉

文例　ON HELP GOSUB 1000

この命令を実行しておくと，以後プログラムの実行中に HELP キーが押されるたびに，指定した開始行から始まる割り込み処理ルーチンに処理が移ります．この命令は割り込みの対象として HELP キーを使っていることを除けば，他の割り込み定義命令(KEY，COM，STOPなど)と同様な使い方で利用できます．

〈行番号〉には，処理ルーチンの開始行番号を指定します．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURNによって行います．たんにRETURNとしたときは，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します．

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないようにしてください．

INPUT実行中に割り込みが生じた場合は，RETURNだけで復帰させるとINPUTの次の命令から実行を再開しますから，行番号を指定するか，プログラムで適切な処理をしなければなりません.

本来この命令は，アプリケーションプログラムを作成する際，プログラム中にプログラムの使い方，入力の仕方などの説明を書いた表示ルーチンを用意しておき，ユーザーがそのプログラムの実行中，使い方がわからなくなったときに HELP キーを押して指示を受ける，といった用途のために用意されているものです.

**注意**：この命令によって割り込み処理ルーチンに分岐させるには，割り込みがあったとき HELP ON の状態でなければなりません.

**参照**：HELP ON／OFF／STOP，RETURN

# ON KEY GOSUB　C

| | |
|---|---|
| 機能 | ファンクションキーによる割り込みルーチンの開始行を定義します. |
| 書式 | ON KEY GOSUB 〈行番号〉〔,〈行番号〉…〕 |
| 文例 | ON KEY GOSUB 100, 200, 300, 400 |

この命令を実行しておくと，以後プログラムの実行中にファンクションキーが押されるたびに，各ファンクションキーに対応する指定開始行から始まる割り込みルーチンに処理が移ります.

〈行番号〉には，割り込みが発生したときに分岐させる処理ルーチンの開始行番号を，ファンクションキーの番号順に並べて指定します.〈行番号〉の並び中で，行番号の指定を省略した場合は，それに対応するファンクションキーによる割り込みは起こりません．したがって，たとえばON KEY GOSUB 100,, 300と指定した場合，f•1 キーおよび f•3 キー以外は割り込みの対象となりません.

ファンクションキーは10個ありますから，最大10個まで〈行番号〉を並べて書くことができます．行番号の代わりにラベル名を使うこともできます.

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURNによって行います．たんにRETURNとしたときは，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します.

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないようにしてください.

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みが発生したとき，KEY　ONの状態でなければなりません．
この命令を実行すると，ファンクションキー本来の機能(ファンクションキーを押すとKEYで定義した文字列が入力される機能)は無視されます．ただし，定義された文字列の内容は，CONSOLEで指定することにより表示することは可能です．

参照：KEY　ON／OFF／STOP，RETURN，サンプルプログラム30

# ON MOUSE GOSUB　Ⓒ

機能　マウスによる割り込みが発生したとき，分岐するサブルーチンの開始行を定義します．

書式　ON　MOUSE(〈事象番号〉)　GOSUB　〈行番号〉

文例　ON　MOUSE(2)　GOSUB　＊MOUSEP1

この命令を実行しておくと，以後プログラムの実行中にマウスの種々の動作が起こるたびに，指定した開始行から始まる割り込み処理ルーチンに処理が移ります．

〈事象番号〉と事象の関係は次のとおりです．

1　：　マウスが移動した
2　：　左側のボタンが押された
3　：　右側のボタンが押された
4　：　左側のボタンが離された
5　：　右側のボタンが離された

〈行番号〉には分岐させる開始行番号を指定します．

処理ルーチンからの復帰は，RETURNで行います．たんにRETURNとしたときには，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します．

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないでください．

割り込み処理ルーチンに制御が移ると，対応する事象に対する割り込みは自動的に停止状態となります．

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みがあったときにMOUSE　ONの状態でなければなりません．

参照：MOUSE，MOUSE(関数)，MOUSE　ON／OFF／STOP

# ON PLAY GOSUB

S C

| | |
|---|---|
| 機　能 | PLAY割り込み処理ルーチンの開始行を定義します. |
| 書　式 | ON　PLAY(〈チャンネル番号〉, 〈残りバイト数〉)　GOSUB　〈行番号〉 |
| 文　例 | ON　PLAY(1, 1024)　GOSUB　＊NEXT.PLAY |

この命令を実行しておくと，バックグラウンド(BASICプログラムの実行と独立して演奏する形態．PLAYのMBコマンド参照）で演奏中に，〈チャンネル番号〉のサウンドバッファ内の未演奏音楽情報が〈残りバイト数〉以下になったとき，指定した開始行から始まる処理ルーチンに処理を移します．

〈チャンネル番号〉に指定する値とその意味の関係は次のとおりです．

| | | |
|---|---|---|
| 0 | ： | 全チャンネルの中で最大の残りバイト数をもつもの |
| 1〜6 | ： | 各チャンネル |
| 負 | ： | 全チャンネルの中で最小の残りバイト数をもつもの |

〈残りバイト数〉は，0〜32767の範囲です．

処理ルーチンからの復帰は，RETURNで行います．たんにRETURNとしたときには，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します．

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないでください．

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みがあったときにPLAY　ONの状態でなければなりません．

PLAYの割り込みは，BASICの他の割り込み命令と異なり，割り込み処理ルーチンに制御が移っても自動的に割り込み停止状態となりません．したがって，処理ルーチンの先頭ではPLAY　OFFまたはPLAY　STOPを実行し，終わりでPLAY　ONを実行する必要があります．

参照：PLAY，PLAY　ON／OFF／STOP，サンプルプログラム41

# ON SRQ GOSUB　G C

機　能　SRQ(サービスリクエスト)の受信による割り込みが発生したとき，分岐する処理ルーチンの開始行を定義します．

書　式　ON　SRQ　GOSUB　〈行番号〉

文　例　ON　SRQ　GOSUB　100

この命令を実行しておくと，コントローラがSRQを受信したときに割り込みが発生し，指定された〈行番号〉から始まる処理ルーチンに分岐します．このサブルーチンには，POLLが含まれている必要があります．

〈行番号〉に0を指定すると，この命令を実行した後，SRQの受信による割り込みが禁止されます．再びSRQの受信による割り込みを許可するためには，もう一度ON　SRQ　GOSUBで行番号を指定しなおす必要があります．なお，SRQの受信による割り込みの禁止／許可を行うには，SRQ　ON，SRQ　OFFを使用することもできます．

処理ルーチンからの復帰は，RETURNで行います．たんにRETURNとしたときには，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します．

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないでください．

SRQを受信してSRQサブルーチンに分岐すると，これ以降のSRQの受信による割り込みは自動的に禁止されます．

注意：この命令によって割り込み処理ルーチンに分岐させるには，割り込みがあったときにSRQ　ONの状態でなければなりません．
　　この命令はコントローラのみが使用できます．

参照：POLL，SRQ　ON／OFF／STOP

# ON STOP GOSUB　C

機　能　STOP キーによる割り込み処理ルーチンの開始行を定義します．

書　式　ON　STOP　GOSUB　〈行番号〉

文　例　ON　STOP　GOSUB　100

この命令を実行しておくと，以後プログラムの実行中に STOP キーが押されるたびに，指定した割り込み処理ルーチンに処理が移ります．この命令は割り込みの対象として STOP キー

を使っていることを除けば，他の割り込み定義命令(KEY, COM, HELPなど)と同様な使い方で利用できます．

〈行番号〉には，処理ルーチンの開始行番号を指定します．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURNによって行います．たんにRETURNとしたときは，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します．

ただし，行番号つきのRETURNでは，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないようにしてください．

注意：この命令によって割り込み処理ルーチンに分岐させるには，STOP ONの状態でなければなりません．
この命令はアプリケーションプログラム実行中に，ユーザーが誤って[STOP]キーを押してしまい，プログラムの実行が中断されてしまうのを防止するためのものです．したがって，不用意にこの命令が実行されてしまうと，本来の[STOP]キーおよび[CTRL]＋[C]の機能は失われ，プログラムの中断ができなくなってしまいます．アプリケーションプログラムの動作が完全になってから，この命令を使うようにしてください．

参照：RETURN，STOP ON／OFF／STOP

# ON TIME$ GOSUB [C]

| | |
|---|---|
| 機能 | **内蔵クロックによる割り込みの発生時刻と，そのとき分岐する処理ルーチンの開始行を定義します．** |
| 書式 | ON TIME$＝"〈時刻〉" GOSUB 〈行番号〉 |
| 文例 | ON TIME$＝"07：30：00" GOSUB ＊MORNINGCALL |

この命令を実行しておくと，以後プログラムの実行中，指定時刻になったときに指定した開始行から始まる割り込み処理ルーチンに処理が移ります．この命令は割り込みの対象として内蔵クロックを使っていることを除けば，他の割り込み定義命令(KEY，COM，HELP，STOPなど)と同様な使い方で利用できます．

〈時刻〉には，割り込み時刻を，"hh：mm：ss"の形の文字列(時：分：秒を各2桁で表したもの．TIME$の設定と同様)で設定します．なお，この命令で制御できる分解能は2秒ですから，±1秒単位の誤差が出ることがあります．

〈行番号〉には，処理ルーチンの開始行番号を指定します．

処理ルーチンからの復帰は，一般のサブルーチンの場合と同じで，RETURN によって行います．たんに RETURN としたときは，中断した所から再開し，後ろに行番号を指定したときは，その行から再開します．

ただし，行番号つきの RETURN では，割り込みが発生した箇所と同じスタックレベルのプログラム行にもどるように指定してください．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないようにしてください．

**注意**：この命令によって割り込み処理ルーチンに分岐させるには，TIME$　ON の状態でなければなりません．
なお，この命令は，実行された時点で，設定された時刻から現在の時刻を引算して割り込み時刻を決定します．したがって，この命令の実行後に TIME$で現在時刻の変更を行うと，割り込み時刻に狂いが生じますので注意してください．

**参照**：RETURN，TIME$，TIME$　ON／OFF／STOP，サンプルプログラム31

# OPEN　Ⓒ

| | |
|---|---|
| 機　能 | ファイルを開きます． |
| 書　式 | OPEN 〈ファイルディスクリプタ〉〔FOR 〈入出力モード〉〕〔〈共用モード〉〕<br>AS 〔#〕〈ファイル番号〉 |
| 文　例 | OPEN "B：DATA1" AS #1<br>OPEN "F1" FOR OUTPUT DENY ALL AS #2<br>OPEN "COM1：N81XN" AS #1 |

この命令は，〈ファイルディスクリプタ〉で指定したファイルをオープンして入出力操作のための専用バッファを用意し，それに〈ファイル番号〉をつけます．以後，ファイルへの入出力は，〈ファイル番号〉を指示することにより行います．

プログラム中で，複数の OPEN を使うことにより，同時に複数のファイルをオープンすることができます．同時にオープンできるファイル数は，最高13ですが，BASIC の起動時に宣言したファイルの数（インタプリタの場合，起動パラメータ／F に指定した値．コンパイラでコンパイルされたプログラムの場合，REM　$FILE で指定した値）を超えてはなりません．複数のファイルをオープンする場合，〈ファイル番号〉には 1 から13までのそれぞれ異なった数値を指定します．

〈入出力モード〉はファイルへのアクセス方法を指定するものです．モードには次の 4 種類があります．

INPUT　：入力モード．既存のファイルから入力を行うよう指示．
OUTPUT　：出力モード．新しくファイルを作り，出力を行うよう指示．
APPEND　：追加モード．既存のファイルの終わりに追加出力を行うよう指示．
省略時　：ランダムモード．ディスクファイルのランダム入出力，または RS-232C 回線ファイルへの入出力を行うよう指示．

FOR〈入出力モード〉を指定した場合，シーケンシャルファイルに対して入出力を行うことを指示します．

FOR〈入出力モード〉を省略した場合，ランダムファイルに対して入出力を行うことを指示します．

〈ファイルディスクリプタ〉に指定したファイルが RS-232C 回線ファイルの場合は，シーケンシャル入出力ですが，ランダム入出力と同様，FOR〈モード〉は省略してください．

ランダム入出力を行うためには，OPEN を実行して用意した専用バッファに対し，FIELD によりフィールド変数を割り当てなければなりません．

〈共用モード〉はファイルの共用制御を行う場合に指定するパラメータで，MS-DOS(Ver 3.1)に MS-NETWORKS が付加されている環境においてのみ指定できるものです．〈共用モード〉には次の5種類があります．

DENY　READ：このファイルがオープンされている間，他プロセス／自プロセスのいかんによらずデータの入力をともなうファイルオープンは禁止されます(以降 OUTPUT モードによる同時オープンのみが可能です．INPUT／APPEND／入出力モード指定なしモードによるオープンは拒否されます)．

DENY　WRITE：このファイルがオープンされている間，他プロセス／自プロセスのいかんによらずデータの書き込みをともなうファイルオープンは禁止されます(以降, INPUT モードによる同時オープンのみが可能です．OUTPUT／APPEND／入出力モード指定なしモードによるオープンは拒否されます)．

DENY　ALL　：このファイルがオープンされている間，他プロセス／自プロセスおよび入出力モードのいかんによらずこのファイルを同時にオープンすることはできません．

DENY　NONE：入出力とも可能です．

省略した時　：このファイルがオープンされている間，他プロセスは入出力モードのいかんによらずこのファイルを同時にオープンすることはできません．自プロセスでは，共用モードが省略されたオープンに限り同時オープンが可能です．

〈共用モード〉はディスクファイルに対してのみ指定可能なパラメータです．

**注意**：INPUT(入力)モード，APPEND(追加)モードでは，指定されたファイルが存在しないと，OPEN 実行時に"File not found"エラーとなります．OUTPUT(出力)モードでは，常に指定された名前のファイルが新しく作られ，同一名のファイルがあった場合にはそのファイルは削除されて，新しく作り直されます．ランダムモードでは，指定されたファイルが存在しないと新たに作られますが，すでに存在しているときは指定ファイルに対して入出力が行われます(OPEN 実行時にファイルの削除や破壊は起こりません)．

RS-232C 回線ファイルをオープンする場合は，〈ファイルディスクリプタ〉として，ファイル名 COM〈回線番号〉に続けて，パリティ，ワード長などを指定する必要があります．指定形式は次のとおりです．

**COM〔〈回線番号〉〕：〔〈パリティ〉〔〈ワード長〉〔〈ストップビット〉〔〈XON スイッチ〉〔〈S パラメータ〉〕〕〕〕〕**

〈回線番号〉には，1，2，3を指定できます．

1　：　第1回線(標準ボード)
2　：　第2回線
3　：　第3回線

2，3は，RS-232C 拡張インタフェースボードがセットされているときのみ指定可能です．〈回線番号〉が省略された場合，第1回線となります．

〈パリティ〉はE，O，N で表し，それぞれ偶数パリティ，奇数パリティ，パリティ無しを意味します．

〈ワード長〉は1文字を表すのに用いるビット数で，7または8で指定します．7を指定した場合には，必ず〈パリティ〉でOまたはEを指定しなければなりません．また，8を指定した場合〈パリティ〉はNでなければなりません．

〈ストップビット〉はストップビット数を決めるもので，1，2，3で指定します．1は1ビット，2は1.5ビット，3は2ビットを意味します．

〈XON スイッチ〉は XON／XOFF による通信制御を行うことを指定します．スイッチとして X が指定された場合 XON／XOFF 制御を行います．N が指定された場合にはこの制御は行われません．

〈S パラメータ〉は〈ワード長〉に7を指定したときに，キャラクタコード 128 以上の文字(カナ文字など)の入出力ができなくなるのを補助するパラメータです．S または N で指定します．S を指定したとき入出力可能で，N を指定したときは入出力不可能になります．

OPEN で扱うことのできるファイル機器は次のとおりです．

| ファイル | 使用できるモード |
|---|---|
| ディスクファイル | INPUT, OUTPUT, APPEND, 省略 |
| RS-232 C コミュニケーションファイル(COM〈ポート番号〉：) | 省略 |
| キーボードファイル(KYBD：) | INPUT |
| スクリーンファイル(SCRN：) | OUTPUT |
| プリンタファイル(LPT：) | OUTPUT |

参照：CLOSE, EOF, FIELD, GET, INPUT #, INPUT$, LOC, LOF, PRINT #, PUT, WRITE #, サンプルプログラム 1, 26, 27, 28, 29

# OPTION BASE　C

機　能　　配列の添字の最小値を指定します．

書　式　　OPTION　BASE { 0 | 1 }

書　式　　OPTION　BASE 1

OPTION　BASE は，配列の添字の最小値を 0 または 1 に指定します．この命令が用いられない場合，最小値は 0 となります．

この命令は，プログラム中で配列変数が宣言，または引用される前に置かなければ効果がありません．

また，添字の最小値を一度指定すると，再指定によって最小値を変更することはできません．この指定を解除，あるいは変更するには，RUN あるいは CLEAR を行わなければなりません．

注意：OPTION　BASE　1 を実行して添字の最小値を 1 に指定した場合，以後，配列の添字として 0 が用いられると，"Subscript　out　of　range" エラーが起こるようになります．また，添字の最小値を再指定しようとすると，"Duplicate　Definition" エラーになります．

参照：DIM，サンプルプログラム 5

# OUT Ⓒ

機 能　出力ポートに1バイトのデータを送出します．

書 式　OUT 〈ポート番号〉，〈式〉

文 例　OUT 64, 32

〈ポート番号〉は出力ポートの番号で，0～32767(&H0～&H7FFF)の範囲内の整数で指定します．

〈式〉はポートに出力するデータで，0～255の範囲内の整数で指定します．

ポート番号と装置との対応については，各機種本体に添付されているハードウェアマニュアルを参照してください．

参照：INP，WAIT

# 〔1〕PAINT Ⓒ

機 能　指定された境界色で囲まれた領域を，指定された色でぬります．

書 式　PAINT ｜(Wx, Wy)｜ 〔, 〈領域色〉〔, 〈境界色〉〕〕
　　　　　　　｜STEP(x, y)｜

文 例　PAINT (−50, 120), 3, 4
　　　　PAINT STEP(10, 10), 6

(Wx，Wy)を中心点として，〈境界色〉で囲まれた領域を，指定された〈領域色〉でぬりつぶします．(Wx，Wy)はワールド座標で指定します．STEPをつけて，LP(最終参照点)からの相対座標(x，y)で指定することもできます．

〈領域色〉，〈境界色〉はともにパレット番号で指定します．〈領域色〉が省略された場合には，COLORで指定されたフォアグラウンドカラーが用いられます．〈境界色〉が省略された場合には，〈領域色〉の指定と同じパレット番号が〈境界色〉として用いられます．

(Wx，Wy)の色が境界色と同じ色であった場合には，PAINTは，何の画面操作も行いません．

注意：PAINTはビューポート内でのみ働きますので，ビューポートの境界はPAINT動作の境界とみなされます．また，(Wx，Wy)がウィンドウの外にある場合には，"Illegal function call" エラーとなります．

参照：〔1〕COLOR，〔2〕PAINT，VIEW，サンプルプログラム17

# 〔2〕PAINT　C

**機　能**　指定された境界色で囲まれた領域を，指定されたタイルパターンで埋めます．

**書　式**　PAINT ｜(Wx, Wy)｜，〈タイルストリング〉〔,〈境界色〉〕
　　　　　　　　｜STEP(x, y)｜

**文　例**　PAINT (255, 120), TILE$, 4

(Wx, Wy)を中心点として，〈境界色〉で囲まれた領域をタイルパターンで埋めます．(Wx, Wy)はワールド座標で指定します．STEP をつけて，LP(最終参照点)からの相対座標(x, y)で指定することもできます．

〈タイルストリング〉は，タイリングに用いられる基本タイルの大きさと模様を決定する文字列です．タイルの大きさは，横方向は 8 ドット分と決められていますが，縦方向の長さは〈タイルストリング〉の文字列の長さで決まります．縦方向が n ドットのタイルを作るためには，〈タイルストリング〉として，白黒モードで n 文字，8 色中・8 色カラーモードで 3*n 文字，4096 色中・16 色カラーモードで 4*n 文字の長さがそれぞれ必要です．

タイルの模様は，〈タイルストリング〉の文字列のキャラクタコードの 2 進数表現(ビットパターン)により表現されます．〈タイルストリング〉の指定の方法は白黒モードとカラーモードとで異なります．

●白黒モードの場合

〈タイルストリング〉中の各 1 文字(バイト)のコードがタイルの横 8 ドットの線に対応します．文字コードの 2 進数表現で，タイルの横 8 ドットのうち，1 に対応するドットはセット(白)，0 に対応するドットはリセット(黒)として表されます．〈タイルストリング〉が n 文字の長さであれば，その文字列の表す模様は，このようにして決定される横 8 ドットのパターンを縦に n 文字分だけ並べたパターンになります．

例)CHR$(&HAA)+CHR$(&H55)

| 16 進数表現 | 2 進数表現 | ドットパターン |
|---|---|---|
| &HAA | 10101010 | ●○●○●○●○ |
| &H55 | 01010101 | ○●○●○●○● |

2 文字からなる簡単なタイルストリングの例です．この 2 文字は 1 であるビットと 0 であるビットが互いに逆になっています．このパターンを基本タイルとして指定された領域を埋めれば，細かい市松模様になりますが，スクリーンのドットが大変細かいため，灰色のように見えます．

●カラーモードの場合

白黒モードと同様に，タイルの模様は〈タイルストリング〉に対応するビットパターンによって決定されます．白黒モードと異なる点は，8色中•8色モードの場合は3文字分ごとに，また4096色中•16色モードの場合は4文字分ごとに，タイルの横8ドット(1ライン分)の色がパレット番号として決定されることです．色の決定のされ方については，次の例を参照してください．

例) 8色中・8色モードあるいは4096色中・8色モードの場合

CHR$(&HAA)＋CHR$(&H55)＋CHR$(&HFF)

| ドット | 1 2 3 4 5 6 7 8 | |
|---|---|---|
| &HAA | 1 0 1 0 1 0 1 0 | ← $2^0$の桁 |
| &H55 | 0 1 0 1 0 1 0 1 | ← $2^1$の桁 |
| &HFF | 1 1 1 1 1 1 1 1 | ← $2^2$の桁 |
| 対応するパレット番号 | 5 6 5 6 5 6 5 6 | |

各ドットについて，縦方向に2進数としてパレット番号に読み換えます．この場合，指定するタイルストリングのうちの最初の文字が最下位ビット($2^0$)の桁を表し，以後，$2^1$の桁，$2^2$の桁，となります．

この例では，パレット番号5とパレット番号6が交互に，横8ドットに指定されたことになります．したがって，パレットが初期状態であれば水色と黄色が交互に出る模様になります．

例)4096色中•16色モードの場合

CHR$(&HCE)＋CHR$(&H8C)＋CHR$(&HD0)＋CHR$(&H22)

| ドット | 1 2 3 4 5 6 7 8 | |
|---|---|---|
| &HCE | 1 1 0 0 1 1 1 0 | ← $2^0$の桁 |
| &H8C | 1 0 0 0 1 1 0 0 | ← $2^1$の桁 |
| &HD0 | 1 1 0 1 0 0 0 0 | ← $2^2$の桁 |
| &H22 | 0 0 1 0 0 0 1 0 | ← $2^3$の桁 |
| 対応するパレット番号 | 7 5 8 4 3 3 9 0 | |

前の例と同じように，各ドットについて，縦方向に2進数としてパレット番号に読み換えることにより，表のような結果となります．

**注意**：カラーモードの場合，〈タイルストリング〉の文字数が3(4096色中•16色モードの場合は4)の倍数でないときには，文字列の後ろから，3(あるいは4)で割った余りの数だけ無視

されます．また，3文字(あるいは4文字)に満たない場合には“Illegal function call”エラーとなります．

すでにタイリングされている領域を異なるパターンでタイリングしようとすると，時間がかかったり，スタックを消費するために“Out of memory”エラーになったりすることがあります．

また，PAINTはビューポート内でのみ働きますので，ビューポートの境界がPAINT動作の境界とみなされます．また，(Wx, Wy)がウィンドウの外にある場合には，“Illegal function call”エラーとなります．

参照：〔1〕PAINT，VIEW

# PCAL$ 関数 D C

機能　パック10進数の演算を行い，その結果を得ます．

書式　1) PCAL$(〈パック10進数1〉,〈算術演算子〉,〈パック10進数2〉〔,〈丸め指定〉〕)

2) PCAL$(ABS,〈パック10進数〉)

3) PCAL$(INT,〈パック10進数〉〔,〈機能〉〕)

文例　A$=PCAL$(J$, *, K$, 1)

B$=PCAL$(ABS, M$)

C$=PCAL$(INT, N$, 1)

書式 1)〈パック10進数1〉と〈パック10進数2〉の四則演算または指数演算を行い，その結果をパック10進数で得ます．

演算の内容は，〈算術演算子〉で指定します．

＋ ：〈パック10進数1〉に〈パック10進数2〉を加えます．

－ ：〈パック10進数1〉から〈パック10進数2〉を減じます．

＊ ：〈パック10進数1〉に〈パック10進数2〉を乗じます．

／ ：〈パック10進数1〉を〈パック10進数2〉で除します．

^ ：〈パック10進数1〉を〈パック10進数2〉乗します．

〈丸め指定〉は，パック10進数の演算結果の有効桁数を超えた部分を，切り捨てるか，または四捨五入するかを指定します．

0または省略 ：19桁目で四捨五入

1 ：19桁目以降を切り捨て

演算の際，パック10進数の最大値または最小値を超えた場合には"Overflow"（OV）エラーとなります．

除算（／）を行う場合に〈パック10進数2〉が0であると，"Division by zero"（／0）エラーとなります．

また，〈パック10進数1〉および〈パック10進数2〉がパック10進数として妥当な内容でない場合には，"Type mismatch" エラーとなります．

書式 2）〈パック10進数〉の絶対値をパック10進数で得ます．

〈パック10進数〉がパック10進数として妥当な内容でない場合には，"Type mismatch" エラーとなります．

書式 3）〈パック10進数〉の整数部分を取り出したものをパック10進数で得ます．

〈機能〉は小数点以下を切り捨てるか，または四捨五入するかを指定します．

| | | |
|---|---|---|
| 0または省略 | ： | 小数点以下第1位を四捨五入 |
| 1 | ： | その値を超えない最大の整数を得る |
| 2 | ： | 小数点以下を無条件に切り捨て |

〈パック10進数〉がパック10進数として妥当な内容でない場合には，"Type mismatch" エラーとなります．

参照：PCHK，PCNV，PCNV$

# PCHK 関数 D C

**機能** パック10進数の大小関係および10進文字列の妥当性を調べます．

**書式** 1）PCHK(〈文字列〉)

2）PCHK(〈パック10進数1〉，〈関係演算子〉，〈パック10進数2〉)

**文例**

```
IF NOT PCHK(A$) THEN *REINPUT
IF PCHK(A$, >, B$) THEN SWAP A$, B$
```

書式 1）〈文字列〉の中に数字，コンマ，ピリオド，スペース，＋，－および指数表記のためのDまたはE以外の文字が含まれている場合には整数値の0（偽）が，それ以外の場合には整数値の－1（真）が，それぞれ関数値として得られます．

空の文字列が指定された場合には"0"とみなされ，関数値は－1（真）となります．

書式 2）〈パック10進数1〉と〈パック10進数2〉の大小関係を調べ，〈関係演算子〉で指定した大小関係を満たしていれば真（－1）が，そうでなければ（0）が，関数値として得ら

れます．〈関係演算子〉には，通常の場合と同様，次のものが指定できます．

| | | |
|---|---|---|
| ＝ | ： | 〈パック10進数1〉と〈パック10進数2〉が等しい |
| ＜ | ： | 〈パック10進数1〉が〈パック10進数2〉よりと小さい |
| ＞ | ： | 〈パック10進数1〉が〈パック10進数2〉よりも大きい |
| ＝＜，＜＝ | ： | 〈パック10進数1〉が〈パック10進数2〉よりも小さいまたは等しい |
| ＝＞，＞＝ | ： | 〈パック10進数1〉が〈パック10進数2〉よりも大きいまたは等しい |
| ＜＞，＞＜ | ： | 〈パック10進数1〉と〈パック10進数2〉が等しくない |

また〈パック10進数1〉，〈パック10進数2〉がパック10進数として妥当でない場合には〝Type mismatch″エラーとなります．

参照：PCAL$，PCNV，PCNV$

## PCNV 関数 　D C

| | |
|---|---|
| 機　能 | パック10進数を倍精度型実数値に変換します． |
| 書　式 | PCNV(パック10進数〉) |
| 文　例 | A#＝PCNV(B$) |

〈パック10進数〉を，倍精度型実数値に変換します．変換結果が倍精度数値の最大値を超える場合には，〝Overflow″(OV)エラーとなります．

また，〈パック10進数〉がパック10進数として妥当な内容でない場合には，〝Type mismatch″エラーとなります．

参照：PCAL$，PCHK，PCNV$

# PCNV$ 関数

**D** **C**

| | |
|---|---|
| 機能 | パック10進数，10進文字列，2進数の相互変換を行います． |
| 書式 | 1）PCNV$(〈10進文字列〉〔, 0〕) |
| | 2）PCNV$(〈パック10進数〉, 1〔, 〈書式制御文字列〉〕) |
| | 3）PCNV$(〈式〉, 2) |
| | 4）PCNV$(〈10進文字列〉, 3, 〈書式制御文字列〉) |
| 文例 | A$=PCNV$("123.45") |
| | PRINT　PCNV$(A$, 1) |
| | A$=PCNV$(123.45, 2) |
| | PRINT　PCNV$(A$, 3,"####.###") |

各種の変換を行います．

**書式 1)** 〈10進文字列〉をパック10進数に変換します．

〈10進文字列〉に数字，コンマ，ピリオド，スペース，＋，－および指数表記のためのDまたはE以外の文字が含まれる場合には〝Illegal function call〞エラーとなります．文字列の途中にあるコンマ，スペースは無視します．文字列が空の場合には"0"とみなします．

**書式 2)** 〈パック10進数〉を10進文字列に変換します．

変換の際，〈書式制御文字列〉を指定すれば，変換結果を編集することができます．〈書式制御文字〉の指定方法は書式 4）を参照してください．

**書式 3)** 〈式〉の値をパック10進数に変換します．

〈式〉は整数型，単精度実数型，倍精度実数型のいずれかでなければなりません．

**書式 4)** 〈10進文字列〉に対して〈書式制御文字列〉で指定された編集処理を施します．

文字列に対してはPRINT　USINGによる数値の編集が行えないため，この関数はそれと等価な編集処理を可能にしたものです．

〈文字列〉に数字，ピリオド，スペース，＋，－および指数表記のためのDまたはE以外の文字が含まれている場合には，〝Illegal function call〞エラーとなります．

〈書式制御文字列〉中に使用できる書式制御文字は，次のとおりです．

**●書式制御文字**

| | |
|---|---|
| # | 数値を出力する桁数を指定します．指定した桁数より数値の桁数が小さいときには右づめで出力されます． |

．(ピリオド)

小数点の位置を指定します．小数点以下の部分で冗長となる行には0が出力されます．

＋　〈書式制御文字〉の最初または最後につけた場合，数値の符号がそれぞれ前または後ろに出力されます．2個以上の〝＋〞を並べた場合には，余分は後述の制御文字以外の文字と同じ扱いとなり，そのまま出力されます．

－(マイナス記号)

〈書式制御文字列〉の最後につけた場合，数値が負の数のときに数値の後ろに〝－〞が出力されます．前につけたり，2個以上並べた場合には後述の制御文字以外の文字扱いとなり，そのまま出力されます．

＊＊　〈書式制御文字列〉の先頭につけた場合，数値領域の左側に空白部ができたとか，そこを〝＊〞で埋めて出力します．この〝＊＊〞は2桁分の領域を確保します．

¥¥　〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に〝¥〞を出力します．〝¥¥〞は2桁分の領域を確保しますが，このうち1桁分は〝¥〞の出力領域として使われます．後述の指数形式の書式指定を行った場合には，正しく出力されないことがあります．

＊＊¥　〈書式制御文字列〉の先頭につけた場合，前記の2つ(＊＊と¥¥)両方の機能となります．〝＊＊¥〞は3桁分の領域を確保しますが，このうち1桁分は〝¥〞の出力領域として使われます．

，(コンマ)

桁数指定の〝#〞の並びの中においた場合，数値の整数部が3桁毎に〝，〞で区切られて出力されます．ただし，前記の〝．〞より右側においた場合は，数値の最後に〝，〞が出力され，3桁毎の区切りは行われません．

^^^^　桁数指定の〝#〞の後ろにつけた場合，指数形式(浮動小数点形式)で出力されます．

●その他の書式制御文字

_(下線)　前述の制御文字1文字をたんに文字として表示するため使用します．〝_〞に続く1文字は常に書式制御機能を持たない文字として出力されます．

●制御文字以外の文字

前述の制御文字以外の文字を指定した場合，数値の前や後ろにそのキャラクタが出力されます．ただし，日本語文字は指定できません．

**●数値の領域を超えた場合**

指定した数値領域より数値の桁数が大きい場合，数値の前に“%”が出力されます．数値を丸めた（四捨五入した）ことによって桁数が領域より大きくなった場合も，丸めた数値の前に“%”が出力されます．

参照：PCAL$，PCHK，PCNV，サンプルプログラム 42

# PEEK 関数 C

| 機能 | メモリ上の指定番地の内容を読み出します． |
|---|---|
| 書式 | PEEK(<アドレス>) |
| 文例 | A=PEEK(&H100) |

指定されたメモリ上の番地から1バイト(8ビット)のデータを読み出します．

<アドレス>は2バイトの値で，0～65535(&H0～&HFFFF)の範囲で指定します．この値はこの命令実行前にDEF　SEGで宣言されたセグメントベースからの相対アドレスを意味します．

なお，各種制御領域のセグメントベースは，SEGPTRによって得ることができます．

参照：CLEAR，DEF　SEG，POKE，SEGPTR，VARPTR

# PLAY S C

| 機能 | 音楽演奏を行います． |
|---|---|
| 書式 | PLAY 〔#<モード番号>,〕〔<文字列 1>〕〔,<文字列 2>〕〔,<文字列 3>〕〔,<文字列 4>〕〔,<文字列 5>〕〔,<文字列 6>〕 |
| 文例 | PLAY #0,"L4O5CDEFG","L4O3GAB>CD" |

<文字列 1～6>で指定するMML（ミュージックマクロランゲージ）の指示に従って，音楽を演奏します．

<文字列 1～3>はチャンネル1～3に対応するFM音源，<文字列 4～6>はチャンネル4～6に対応するSSG音源で，合わせて同時に6音までの同時演奏が可能です．

#<モード番号>はチャンネル3のFM音源に対しどのようなモードで演奏するかを指定するものです．指定する値とモードの関係は次のとおりです．

#0　：　楽音モード（通常の演奏に適する）

#1　：　効果音モード（効果音に適する）

#2　：　CSM モード（複合正弦波合成モード）

各モードの詳細については，サウンドボードに添付されている「サウンドボードユーザーズマニュアル」を参照してください．

〈文字列 1～6〉にはそれぞれ，MML のコマンドを文字列で指定します．MML には次のようなコマンドがあります．

Cx, Dx, Ex, Fx, Gx, Ax, Bx
　音程をアルファベットで指定します．音長の指定は，各音程のあとの x（音長数字）で行います．数字の意味は Lx コマンド（後述）と同じで，4 は 4 分音符を表します．このコマンドで音長の指定を省略した場合は，Lx コマンドにより指定された音長をとります．

Mx　SSG 音源（チャンネル 4～6）のエンベロープ周期を設定します．x の値は 1～65535 の範囲になくてはいけません．初期値は M255 です．
　x に指定する値は，発生しようとするエンベロープ周期に対して，次の式により求められます．

　x=667＊T／256　　　T：エンベロープ周期（ms）
　　　　　　　　　　　x：設定値

Sx　SSG 音源（チャンネル 4～6）のエンベロープ形状を指定します．x の値は 1～15 です．
　x の値とエンベロープ形状の対応は次の図のとおりです．初期値は S1 です．

| x | エンベロープ形状 | x | エンベロープ形状 |
|---|---|---|---|
| 0,1,2,3,9 |  | 11 |  |
| 4,5,6,7,15 |  | 12 |  |
| 8 |  | 13 |  |
| 10 |  | 14 |  |

　SSG 音源には固定音量出力モードと Mx コマンド，Sx コマンドで指定される可変出力モードがありますが，各チャンネルを可変出力モードにするのは Sx コマンドです．Mx コマンドは周期を設定するだけでモードは変えません．また，固定音量出力モードにするのは，後述する Vx コマンドです．

Vx　各チャンネルに対して大まかに音量を設定します．
　"V0"が最小で"V15"が最大音量です．SSG 音源に対する初期値は"V7"です．FM 音源に

ついては各音色ごとに最も適した音量が設定されるので初期値は存在しません．また，FM 音源については音量の大小により音色も多少変化します．@x コマンドで音色番号を切り換えると，@Vx コマンド，Vx コマンドによる設定は無効になります．

Lx　このコマンドで音の長さを指定すると，Cx，Dx，Ex，Fx，Gx，Ax，Bx コマンドで音長を省略したときには，すべてこの長さになります．
x は 1〜64 の範囲で，1 は全音符，8 は 8 分音符といったように指定します．このコマンドによる設定を省略した場合の初期値は，"L4"（4 分音符）です．

Qx　音長により定められる長さ（ステップタイム）と，その間で実際に音を出している長さ(ゲートタイム)の比を設定します．x の値は 1〜8 の範囲で，ゲートタイム／ステップタイムの比は x／8 で求められます．初期値は"Q7"です．

Ox　オクターブ値を設定します．x の値は 1〜8 で，数値が大きいほどオクターブは高くなります．初期値は"O4"です．

<,>　現在のオクターブを 1 オクターブだけ上げ（>），下げ（<）します．

Kx　x で指定された高さの音を発生します．x（キー番号）と実際に発生される音程の対応は次のとおりです．ただし，FM 音源においては"O9C"は"O1C"と同じになります．

| オクターブ / 音程 | O1 | O2 | O3 | O4 | O5 | O6 | O7 | O8 | O9 |
|---|---|---|---|---|---|---|---|---|---|
| C | 0 | 12 | 24 | 36 | 48 | 60 | 72 | 84 | 96 |
| C# | 1 | 13 | 25 | 37 | 49 | 61 | 73 | 85 | |
| D | 2 | 14 | 26 | 38 | 50 | 62 | 74 | 86 | |
| D# | 3 | 15 | 27 | 39 | 51 | 63 | 75 | 87 | |
| E | 4 | 16 | 28 | 40 | 52 | 64 | 76 | 88 | |
| F | 5 | 17 | 29 | 41 | 53 | 65 | 77 | 89 | |
| F# | 6 | 18 | 30 | 42 | 54 | 66 | 78 | 90 | |
| G | 7 | 19 | 31 | 43 | 55 | 67 | 79 | 91 | |
| G# | 8 | 20 | 32 | 44 | 56 | 68 | 80 | 92 | |
| A | 9 | 21 | 33 | 45 | 57 | 69 | 81 | 93 | |
| A# | 10 | 22 | 34 | 46 | 58 | 70 | 82 | 94 | |
| B | 11 | 23 | 35 | 47 | 59 | 71 | 83 | 95 | |

Tx　演奏するテンポを指定します．どのチャンネルでテンポを指定してもかまいません．ただし，全チャンネルを通してテンポはただ 1 つしか指定できないので，最後に指定した

チャンネルのテンポが有効となります．x の範囲は 32〜255 までで，"T120"が初期値です．T120 は"♩=120"に相当します．

Rx, Px

休符です．x の値は 1〜64 で，音長を指定します．1 が全休符，4 が四分休符といったように指定します．

+(#), −

いずれも音程(Cx, Dx, Ex, Fx, Gx, Ax, Bx)のあとにつけて半音階を表します．"+"または"#"で半音上げ，"−"で半音下げます．

.（ピリオド）

音程 (Cx, Dx, Ex, Fx, Gx, Ax, Bx) に指定する音長数字のあとにつけることによって，符点音符を表します．休符 (Rx, Px) のあとにつけて符点休符とすることもできます．

^ 前後の音長をつないで 1 つにします．たとえば，"C8^8"のように表現される音は"C4"と等しくなります．

{ … } x

連符を表します．x で指定された音長を {} 内の音符の個数で等分します．ただし，{} 内で音長を指定した場合は正しい連符となりません．また，連符の中に連符や間接指定を含めることもできません．

@x x で示される音色番号の音に音色を切り換えます．

x の範囲は 0〜81 です．このコマンドは，FM 音源チャンネルについてのみ有効です．音色番号と音色の対応は，初期状態では次のようになっています．音色番号の初期値は 0 番です．

内 蔵 基 本 音 色

| 音色番号 | 音 色 名 | 説 明 | 備 考 |
|---|---|---|---|
| 0 | DEFAULT | (HARPSIC-11 番と同じ) | |
| 1 | BRASS 2 | 低域の金管楽器 | |
| 2 | STRING 2 | 高域の弦楽器 | |
| 3 | EPIANO 3 | 丸みを持ったエレクトリックピアノ | |
| 4 | EBASS 2 | シンセサイザベース | |
| 5 | EORGAN 1 | 堅めのエレクトリックオルガン | |
| 6 | PORGAN 1 | 低域のパイプオルガン | |
| 7 | FLUTE | フルート | O5 が最適です |

| 音色番号 | 音 色 名 | 説 明 | 備 考 |
|---|---|---|---|
| 8 | OBOE | オーボエ | |
| 9 | CLARINET | クラリネット | |
| 10 | VIBRAPHON | ビブラフォン | |
| 11 | HARPSIC | ハープシコード | |
| 12 | BELL | ベル | |
| 13 | PIANO | アコースティックピアノ | |
| 14 | MUSHI | 虫の鳴き声 | |
| 15 | DESCENT | 下降するような効果音 | |
| 16 | UFO | UFO が飛んでいるような効果音 | |
| 17 | GRANPRI | レーシングカーが走って行くような効果音 | |
| 18 | LASER 1 | SF 的な効果音 1 | |
| 19 | LASER 2 | SF 的な効果音 2 | |
| 20 | SINWAVE | 正弦波 | |
| 21 | BRASS 1 | 高域の金管楽器 | |
| 22 | BRASS 1 | 低域の金管楽器 | |
| 23 | TRUMPET | トランペット | O 5 が最適です |
| 24 | STRING 1 | 低域の弦楽器 | |
| 25 | STRING 2 | 高域の弦楽器 | |
| 26 | EPIANO 1 | エレクトリックピアノ | |
| 27 | EPIANO 2 | 堅めのエレクトリックピアノ | |
| 28 | EPIANO 3 | 丸みを持ったエレクトリックピアノ | |
| 29 | GUITAR | エレキギター | O5 が最適です |
| 30 | EBASS 1 | シンセサイザベース | |
| 31 | EBASS 2 | ウッドベースに近い音 | O3 が最適です |
| 32 | EORGAN 1 | 堅めのエレクトリックオルガン | |
| 33 | EORGAN 2 | やや丸いエレクトリックオルガン | |
| 34 | PORGAN 1 | 低域のパイプオルガン | |
| 35 | PORGAN 2 | 高域のパイプオルガン | |
| 36 | FLUTE | フルート | O5 が最適です |
| 37 | PICCOLO | ピッコロ | |
| 38 | OBOE | オーボエ | |
| 39 | CLARINET | クラリネット | |
| 40 | GROKEN | グロッケン | |
| 41 | VIBRAPHON | ビブラフォン | |
| 42 | XYLOPHN | シロフォン | |
| 43 | KOTO | 琴 | |
| 44 | ZITAR | チター | |

| 音色番号 | 音 色 名 | 説 明 | 備 考 |
|---|---|---|---|
| 45 | CLAVINET | クラビネット | |
| 46 | HARPSIC | ハープシコード | |
| 47 | BELL | ベル | |
| 48 | HARP | ハープ | O5 が最適です |
| 49 | BELL／BRASS | ベルと金管楽器のユニゾン | |
| 50 | HARMONICA | ハーモニカ | |
| 51 | STEELDRUM | スチールドラム | |
| 52 | TIMPANI | ティンパニー | O3 が最適です |
| 53 | TRAIS | 汽笛の音 | |
| 54 | AMBULANCE | 救急車の音 | 長めの音が適します |
| 55 | TWEET | 小鳥の声 | O6，Q4 が最適です |
| 56 | RAIN DROP | 雨だれの音 | 長めの音が適します |
| 57 | HORN | ホルン | |
| 58 | SNARE DRUM | スネアドラム | |
| 59 | COW BELL | カウベル | |
| 60 | PERC 1 | バネのはじけるような音 | |
| 61 | PERC 2 | 水の入ったグラスをはじくような音 | O5 が最適です |
| 62 | PERC 3 | ウインドチャイムのような音 | O6 が最適です |
| 63 | MUSIC BOX | オルゴールのような音 | |
| 64 | CELLO | チェロ | |
| 65 | LOW BRASS | チューバのような低域の金管楽器 | |
| 66 | WW ENSMBL | 木管楽器のアンサンブル | |
| 67 | AC GUITAR | アコースティックギター | |
| 68 | FLUTE/HARP | フルートとハープのユニゾン | |
| 69 | FUNK PLUC | 打楽器の音 | |
| 70 | FUNK BASS | ファンキーなベースの音 | |
| 71 | SYN LEAD | メロディーライン向けのリードに適した音 | |
| 72 | METAL CLES | 金属的な効果音 1 | O3 が適します |
| 73 | STAIN | 金属的な効果音 2 | O3 が適します |
| 74 | CUBIN GLASS | 氷がグラスの中を転がるような音 | O5 が適します |
| 75 | HUMAN | 人の声のような音 | |
| 76 | WOOD BASS | ウッドベース | O3 が最適です |
| 77 | CHIMES | チャイム | O5 が最適です |
| 78 | SPACY | 機械的な口笛の音 | O5 が最適です |
| 79 | OBOE/B.CLAR | オーボエとバスクラリネットのユニゾン | |
| 80 | OLD STRING | 古いレコードのバイオリンのような音 | |
| 81 | STEEL'S CRY | 金属的な泣き声のような音 | |

備考の欄に特に記入のないものは，どのような音域でもよいか，またはデフォルトが最も適していることを示します．

Yr，d

シンセサイザ LSI のレジスタに直接値を設定します．VOICE　REG と同じ機能を行うものです．

r はレジスタ番号で 0～178，d はレジスタ内容で 0～255 の範囲です．定義されていないレジスタに対するこのコマンドの動作の保証はありません．また，サウンド拡張命令がハードウェアの制御のために使用しているレジスタについて書き込みを行った場合も，動作の保証はありませんので注意してください．

@Vx　FM 音源に対する音量を細かく設定します．

x は 0～127 の範囲で，0 が最小音量，127 で最大音量となります．

@Wx　x で指定された長さだけ何もしないで待ちます．x は 1～64 の範囲で，1 は全音符，8 は 8 分音符といったように指定します．

@Wx コマンドは Yr, d コマンドなどで直接音を発生させる場合に有効です（Rx, Px コマンドでは，一旦発音を停止させてしまいますので音がとぎれてしまいます）．普通の音符（Cx，Dx，Ex，Fx，Gx，Ax，Bx）のあとに@Wx がきた場合は，Rx，Px コマンドと同等です．

_x　移調を指示します．x は音程で，A，B，C…のように指定します．x によって示される音程を，C の音とみなして以降の MML を移調します．各チャンネルごとに独立して指定できます．ただし，Kx コマンドによる指定は移調しません．

移調は次の規則により行われます．

- 音程（A，B，C，D…）をキー番号（0～96）に直します．
- その値に，次表に示すようなベース値を加えて対応するキー番号の音を発生します．

| 指定調 | G♭ | G | A♭ | A | B♭ | B | C | C♯ | D | D♯ | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ベース値 | −6 | −5 | −4 | −3 | −2 | −1 | 0 | 1 | 2 | 3 | 4 | 5 |

たとえば，　PLAY　"__DDEFG"と指定した場合，実際に発生される音は，PLAY　"EF＃GA"と等しくなります．

!　LFO 効果（ビブラート，トレモロ）を加えます．

音色番号指定直後や，VOICE　LFO 実行直後は，LFO 効果が加えられています．ただし，音色データの値によって，LFO 効果がかからない場合があります．

＊　!コマンドの逆で，LOF 効果を切ります．

**Zp, v** 指定チャンネルの音色パラメータの内容を設定します.

pはパラメータ番号で, 0～49の範囲です. また, vはパラメータの内容で 0～16383までの値です. pのパラメータ番号は, VOICEやVOICE COPYにおける音色用配列の添字に, 次のように対応します.

p=(第1添字)+(第2添字)*5

たとえば, 音色用配列 (3, 1) に対応するパラメータ番号は, 次のようになります.

3+1*5=8

vの値については, 各音色パラメータの有効範囲外の値を設定しようとした場合, 動作は保証されません. また, 負の値は8ビットの符号付数値に直して指定してください. たとえば, −5を指定したいときは, 次のように256を加えた値を指定します.

256+(−5)=251

**MF, MB**

PLAYの演奏形態を指定します.

"MF"を指定すると, PLAYによる演奏が終了してから次の命令の実行へ移ります. これをフォアグラウンド指定といいます. 初期状態は"MF"になっています.

"MB"を指定すると, PLAYの音楽情報をサウンドバッファに格納してすぐに次の命令に移ります. これをバックグラウンド指定といいます. "MF", "MB"ともに各チャンネルごとに指定できますが, "MF"で指定されたチャンネルと"MB"で指定されたチャンネルが同時にあった場合, "MF"指定のチャンネルが終了するまで次の命令へは移れません.

MBが指定された場合であっても, 次の条件が起こると演奏は中断されます.

- STOP キーまたはSTOPの実行
- PLAY ALLOCの実行
- エラーの発生による中断

## MMLの間接指定

MMLの一部を文字変数や数値変数で置き換えることも可能です. このことを間接指定といいます. 間接指定には, 次のものがあります.

- **数値の置き換え**

"L=A;"のように"="+(変数名)+";"で表します. ただし, 変数名は配列の要素であったり型宣言文字 ("!" や "%" など) がついていてはいけません.

- **文字列の置き換え**

"X=A$;"のように"X="+(文字変数名)+";"で表します. ただし, 変数名は配列の要素であってはなりません.

なお，コンパイラにおいてはMMLの間接指定を行うことはできません．

注意：PLAYを使用する際には，次のことに注意してください．

(1) PLAYを実行する前には，必ずPLAY　ALLOCを実行して，サウンドバッファを各チャンネルのために用意しておかなければなりません．
(2) エラー発生時には音色は自動的に0番に設定されます．
(3) PLAYにより演奏される音長は他の処理などの影響により若干長くなることがあります．
(4) MBによる演奏時，BASICプログラムの実行は若干遅くなります．
(5) MFによる演奏時に連続してPLAYを実行させた場合，命令解析の時間のためにMMLの最後の音長が長めになったり，演奏が息つぎをしているように聞こえることがあります．
(6) 音量が大きすぎると音がひずむおそれがあります．また，2チャンネル以上を使って演奏を行う場合，1チャンネルで演奏するときに比べ音量が増し，音がひずむ場合があります．このようなときは，@Vxコマンドで音量を調節してください．

参照：VOICE，VOICE　COPY，VOICE　REG，PLAY　ALLOC，サンプルプログラム41

# PLAY ALLOC　S C

| 機能 | サウンドバッファの確保および初期化をします． |
|---|---|
| 書式 | PLAY　ALLOC〔<CH1バッファサイズ>〕〔，<CH2バッファサイズ>〕〔，<CH3バッファサイズ>〕〔，<CH4バッファサイズ>〕〔，<CH5バッファサイズ>〕〔，<CH6バッファサイズ>〕 |
| 文例 | PLAY　ALLOC　255，255，255 |

音楽演奏のためのサウンドバッファを，<CH1バッファサイズ>～<H6バッファサイズ>で指定されたバイト数分，各チャンネル毎に確保します．省略されたチャンネルのバッファサイズは0となります．

PLAY　ALLOCはサウンド拡張命令を使用する前に必ず一度実行しなければなりません．確保しようとするバッファサイズ（各チャンネルの合計値）は必ずCLEARで確保した機械語プログラム領域より小さくなければなりません．

また，PLAY　ALLOCはサウンド拡張機能を初期化します．初期値は次のとおりです．

| | | |
|---|---|---|
| SSGエンベロープ | ： | M255，S1（ただし，固定出力にされている） |
| SSG音量 | ： | V7 |
| 音長 | ： | L4（R4） |

| | | |
|---|---|---|
| 音の長さの割合 | : | Q7 |
| オクターブ | : | O4 |
| テンポ | : | T120 |
| FMチャンネル音色 | : | 0（初期状態の音色） |
| 音色バンク | : | BASIC立ち上げ時の状態に設定 |

参照：PLAY，サンプルプログラム41

# PLAY ON／OFF／STOP　S C

機　能　PLAY割り込みの許可，禁止，停止を制御します．

書　式　1）PLAY　ON
2）PLAY　OFF
3）PLAY　STOP

文　例　PLAY　ON

書式 1）PLAY割り込みを，この命令の次の命令を実行した後，許可します．その後，ON PLAY　GOSUBで定義した条件が満たされるごとに割り込みが発生し，ON PLAY　GOSUBで定義されている処理ルーチンへ分岐します．

書式 2）PLAY割り込みを禁止します．この命令が実行されると，割り込み条件が満たされても割り込みは起こらず処理ルーチンには分岐しません．

書式 3）PLAY割り込みを停止します．この命令が実行されると，割り込み条件が満たされても割り込みは起こらず処理ルーチンには分岐しません．しかし，割り込みのあったことを覚えているため，後でPLAY　ONによって割り込みが許可されると，処理ルーチンに分岐します．

注意：PLAYの割り込みは，BASICの他の割り込み命令と異なり，割り込み処理ルーチンに制御が移っても自動的に割り込み停止状態となりません．したがって，処理ルーチンの先頭ではPLAY　OFFまたはPLAY　STOPを実行し，終わりでPLAY　ONを実行する必要があります．

参照：ON　PLAY　GOSUB，PLAY，サンプルプログラム41

# POINT　C

機　能　LP(最終参照点)を変更します.

書　式　POINT | (Wx, Wy) |
　　　　　　　| STEP(x, y) |

文　例　POINT　(−200, 30)
　　　　POINT　STEP(20, 20)

最後にグラフィック操作の行われた座標を「最終参照点」(Last Referenced Point. 略して LP)といいます. LP は相対座標により座標指定を行う際の基点となります. グラフィック関係の命令はほとんど, 実行することにより, 描画と同時にこの LP も変更します.

POINT はワールド座標(Wx, Wy)で LP をたんに設定, あるいは変更する働きをします. 実際に図形を描いたり, 図形に変化を与えることはありません.

STEP をつけると前の LP からの相対座標による指定となります.

例 1) LINE(−200, 30)−(100, 120), 3
例 2) POINT(−200, 30)
　　　LINE −STEP(300, 90), 3

例1と例2は, まったく同じ図形を描きます.

参照 : LINE, 〔1〕POINT(関数), サンプルプログラム17

# 〔1〕POINT 関数　C

機　能　LP(最終参照点)の値を得ます.

書　式　POINT(〈機能〉)

文　例　WX=POINT(0)

最後にグラフィック操作の行われた座標(最終参照点. 略して LP)を, 〈機能〉に指定する値により, スクリーン座標あるいはワールド座標として得ます.

〈機能〉には 0〜3 の値を指定します.

0 : ワールド座標系の値に変換された LP の X 座標
1 : ワールド座標系の値に変換された LP の Y 座標
2 : スクリーン座標系の値に変換された LP の X 座標
3 : スクリーン座標系の値に変換された LP の Y 座標

LP がウィンドウの外であった場合，〔1〕POINT によって得られる座標に対応する点は，画面上に表示されていない点であることに注意してください．またこのとき，得られた値がスクリーン座標系に変換された座標値であれば，ビューポート外になることに注意してください．

参照：POINT

## 〔2〕POINT 関数 C

| | |
|---|---|
| 機能 | スクリーン座標系の指定された座標に表示されているドットの色を得ます． |
| 書式 | POINT(Sx, Sy) |
| 文例 | COL=POINT(473, 120) |

スクリーン座標系の(Sx, Sy)で指定された座標に表示されているドットの色を，パレット番号で得ます．(Sx, Sy)がビューポートの外にある場合，〔2〕POINT の値は−1となります．

白黒モードの場合，ドットの色はリセット(0：黒を表す)またはセット(1：白を表す)で表され，〔2〕POINT で得られる値はこのいずれかとなります．

参照：〔2〕COLOR，VIEW

## POKE C

| | |
|---|---|
| 機能 | メモリ上の指定番地へデータを書き込みます． |
| 書式 | POKE 〈アドレス〉,〈式〉 |
| 文例 | POKE &HF000, &HFF |

指定されたメモリ上の番地に1バイト(8ビット)のデータを書き込みます．

〈アドレス〉は2バイトの値で，0〜65535(&H0〜&HFFFF)の範囲で指定します．この値はこの命令実行前に DEF SEG で宣言されたセグメントベースからの相対アドレスを意味します．なお，各種制御領域のセグメントベースは，SEGPTR により得ることができます．

〈式〉は書き込まれるデータで，0〜255(&H0〜&HFF)の値でなければなりません．もし小数点以下があると整数化(小数点以下を四捨五入)してから実行されます．

注意：この命令は，現在のメモリの内容を書き換えてしまうため，不用意に使うと BASIC が使っている作業領域を壊してしまい，誤動作の原因となることもあります．使うときには，メモリマップなどで使用可能な領域かどうかを確認してください．

参照：DEF SEG，PEEK，SEGPTR，VARPTR

# POLL

G C

| | |
|---|---|
| 機　能 | シリアルポールを行います. |
| 書　式 | POLL 〈トーカアドレス〉, 〈数値変数〉〔; 〈トーカアドレス〉, 〈数値変数〉〕 |
| 文　例 | POLL 1, B |

POLLが実行されると，まずUNL (アンリスン) コマンドに続き，SPE (シリアルポールイネーブル) コマンドが送出されます.

その後，指定された〈トーカアドレス〉が出力された後，ATN (アテンション) がfalseとなり，指定されたトーカから送信されてくるデバイスステータスを受信し，〈数値変数〉に代入します.

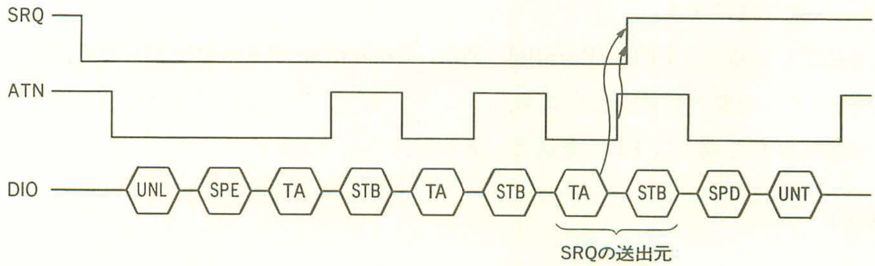
このとき，受信したデバイスステータスのSRQ (サービスリクエスト) ビット (ビット6) がセットされていたならば，この装置がサービスリクエストを送出したと判断します.

サービスリクエストを送出した装置が発見されると，シリアルポールは終了し，それ以降の数値変数にはすべて0が代入されます.

また，これらの情報は，次のようにIEEE (関数) にも代入されます.

IEEE(4)　サービスリクエストを送出した装置のデバイスステータス
IEEE(5)　サービスリクエストを送出した装置のトーカアドレス
IEEE(6)　サービスリクエストに応答しない装置のトーカアドレス

指定された装置の中に，サービスリクエストを送出しているものがない場合，つまり受信したデバイスステータスのSRQビットが，いずれもセットされていなかった場合，IEEE(4) およびIEEE(5) には0が代入され，各〈数値変数〉には，各装置から送出されたデバイスステータスが代入されます.




シリアルポールのタイミング

**注意**：この命令はコントローラのみが使用できます.

**参照**：IEEE

# POS 関数 C

| 機能 | テキスト画面上の現在のカーソルの桁位置を得ます． |
|---|---|
| 書式 | POS(<数式>) |
| 文例 | CX=POS(0) |

現在のカーソルの桁(水平)位置をキャラクタ座標で得ます．値の範囲は0～79(または39)となります．得られる値は，0が画面の左端の桁を，79(または39)が右端の桁を，それぞれ意味します．

<数式>はダミーであり，どんな数式を指定しても結果に変わりはありませんが，省略することはできません．

参照：CSRLIN，サンプルプログラム10

# PPOLL G C

| 機能 | パラレルポールの応答出力用ラインの割り付けを行います． |
|---|---|
| 書式 | PPOLL 〔PPU〕〔，(<リスナアドレス1>，<ppe1／ppd>)〔，(<リスナアドレス2>，<ppe2／ppd>)〕〕 |
| 文例 | PPOLL PPU，1，&H66，2，&H6F |

マスタモードにおいてのみ使用できる命令です．

<リスナアドレス1>で示されるリスナに，<ppe1／ppd>で指定した応答出力ラインを割り付けます．もし指定されれば，同様に，<リスナアドレス2>のリスナに<ppe1／ppd>の応答出力ラインを割り付けます．

PPUを指定した場合，PPU (Parallel Poll Unconfigureメッセージ) を送出したのち，応答出力用ラインの割り付けを行います．

PPUが省略された場合はPPUを送出しません．

参照：IEEE

# PRESET

Ⓒ

**機　能**　画面上の任意の座標のドットを消去します．

**書　式**　PRESET ｜(Wx, Wy)｜〔,〈パレット番号〉〕
　　　　　　　　　 ｜STEP(x, y)｜

**文　例**　PRESET　(100, 100)

ワールド座標(Wx, Wy)で示されるドットを消去します．

(Wx, Wy)の代わりに，STEPをつけて相対座標(x, y)で指定することもできます．

この命令は，オプションの〈パレット番号〉をつけない方が一般的な使い方です．その場合，(Wx, Wy)で表される座標のドットを，現在〔1〕COLORによって設定されているバックグラウンドカラーでぬり変えます(すなわちドットが消える)．

〈パレット番号〉を指定した場合は，PSETとまったく同じに機能し，そのパレット番号のカラーでドットを表示します．

PRESETを実行すると，LP(最終参照点)は，指定した座標点に変更されます．

参照：〔1〕COLOR，PSET，サンプルプログラム17

# PRINT

Ⓒ

**機　能**　画面にデータを出力します．

**書　式**　｜PRINT｜〔〈式〉〕〔｜,｜〈式〉…〕〔｜,｜〕
　　　　　　｜　?　｜　　　　　｜;｜　　　　　｜;｜

**文　例**　PRINT　"COLUMN="; X, "LINE="; Y

〈式〉に，数値や文字列を指定すると，その数値，文字列がそのまま出力され，数値変数や文字変数が指定されると，それらの変数に代入されている数値や文字列が出力されます．〈式〉が省略された場合には改行のみを行います．

数値を出力する場合，その後ろに必ず空白が入ります．また，数値の前には符号用の桁が用意されていて，プラスのときには空白，マイナスのときにはマイナス符号が表示されます．

複数のデータを表示する場合，区切り記号として，コンマ(,)，セミコロン(;)，空白を使います．データを表示する領域は，あらかじめ各行を14文字ごとに分割して定められており，区切り記号によって表示のしかたが変わります．

コンマ(,)を使った場合は，次の領域から表示し，セミコロン(;)あるいは空白を用いた場合には，直前に出力したもののすぐ後ろに出力します．

ダブルクォーテーション(")で囲まれた文字列の前後に，他の文字列や変数を並べる場合，その間の区切り記号を省略することができます．このときは，セミコロンと同様の働きをします．

〈式〉の最後にセミコロンやコンマを指定すると改行が起こらず，その行で次のPRINTによる出力を続行します．

PRINTの短縮形として疑問符(?)を使うことができます．プログラム内で"?"を用いた場合，後でLISTを行うと，自動的に"PRINT"に変換されます．

**注意**：単精度の数値で，指数形式(浮動小数点形式)でなくても6桁以下の桁数で精度に影響をおよぼさず表示できるものは，通常の小数点形式(固定小数点形式)で表示されます．同様に倍精度の数値で16桁以下の桁数で精度に影響をおよぼさず表示できるものは，固定小数点形式の表示になります．また，表示する数値の長さが現在のカーソルの位置より後方にとれない場合(それを表示すると次の行にわたってしまう場合)は改行してからそれを表示します．

**参照**：LPRINT，PRINT USING，サンプルプログラム12

# PRINT # Ⓒ

**機能**　ファイルにデータを書き出します．

**書式**　PRINT #〈ファイル番号〉，〈式〉〔{，|；}〈式〉…〕〔{，|；}〕

**文例**

```
PRINT #2, A ; B ; C
PRINT #1, A$ ; "," ; B$
```

出力モードでオープンしたファイル(シーケンシャルファイル，スクリーンファイル，プリンタなど)や，RS-232C回線ファイルに，〈式〉により指定した文字列，数値などのデータを書き出します．

〈ファイル番号〉には，OPENによって，そのファイルをオープンしたときに使った番号を指定します．

〈式〉には，ファイルに書き込む数値式，文字式を指定します．

PRINT #は，PRINTで画面に出力する場合と同様の形式で，ファイルにデータを書き出しますが，とくにシーケンシャルファイルへの書き出しの際には，入力時にこれらのデータがファイルから正しく読み出せるような形式になるように，適切に区切らなければなりません(INPUT #参照)．

以下，ディスク上のシーケンシャルファイルに対して書き出しを行う場合につき，データの区切りかたについて説明します．

〈式〉が数値式の場合は，セミコロン(；)で区切ります．たとえば，

```
A=123
B=456
C=-78
PRINT  #1, A ; B ; C
```

とすると，ディスクには次のように書き込まれます．

□123□□456□-78　　　　　　(□は空白を意味します)

もし，区切り記号にコンマを使うとプリント領域の間に挿入される余分な空白もディスクに書き出してしまいます(挿入される空白の個数はPRINTの場合と同じです)ので，ディスクの記憶容量がむだ使いされます．

〈式〉が文字式の場合，セミコロンで区切り，かつPRINT #中に独立した区切り記号を入れます(コンマをダブルクォーテーションで囲むなど)．たとえば，

```
A$="CAMERA"
B$="93604-1"
PRINT  #1, A$ ; "," ; B$
```

とすると，ディスクには次のように書き出されます．

CAMERA, 93604-1

","を使わないで，PRINT　#1，A$；B$とした場合には，ディスクにはCAMERA93604-1と書き出されてしまいます．これでは区切りの記号がありませんから2つの別の文字列として読み込むことはできません．

文字式それ自身がデータとしてコンマ，セミコロン，意味のある始めの空白，キャリッジリターン(CHR$(13))，またはラインフィード(CHR$(10))などを含む場合は，ダブルクォーテーションコード(CHR$(34))によって囲んでディスクに書き出さなければなりません．たとえば，

```
A$="CAMERA, □AUTOMATIC"
B$="□□□93604-1"
PRINT  #1, A$ ; "," ; B$
```

とするとディスクには，

CAMERA, □AUTOMATIC, □□□93604-1

と書き出されます．これではA$の中のコンマがデータの区切りとなってしまい，INPUT　#1,

A$, B$としてデータを読み込むと, A$にはCAMERAを, B$には□AUTOMATICを読み込んでしまい, □□□93604-1は読み込まれないで残ってしまいます.

これを避けるには,

PRINT　#1, CHR$(34) ; A$ ; CHR$(34) ; CHR$(34) ; B$ ; CHR$(34)

と指定します. この場合ディスクには,

"CAMERA, □AUTOMATIC""□□□93604-1"

と書き出されますので, INPUT　#1, A$, B$とすれば, "CAMERA, □AUTOMATIC"をA$に, "□□□93604-1"をB$に読み込むことができます.

**注意**: 出力対象ファイルが "SCRN:" の場合, 日本語データの出力はできません.

**参照**: INPUT#, PRINT#　USING, WRITE#, サンプルプログラム28

# PRINT@

G C

**機　能**　データをASCII文字列として送出します.

**書　式**　1) PRINT@　〔〈リスナアドレス〉〕〔,〔〈リスナアドレス〉〕〕; 〈データ〉〔, 〈データ〉…〕〔@〕

2) PRINT@ ; 〈データ〉〔, 〈データ〉…〕〔@〕

**文　例**　PRINT@1, 3 ; A$, B$

PRINT@ ; A, C$

**書式 1)** マスタモードの場合に使用します.

ATN(アテンション)をtrueにしてUNL(アンリスン)コマンド, MTA(マイトークアドレス), リスナアドレスを順次送出します. その後, ATNをfalseにして, データをASCII文字列として送出します.

データが複数個ある場合は"," で区切られて送出され, 最後のデータの後にデリミタが送出されます. デリミタについてはCMD　DELIMを参照してください. 命令の最後に"@"がある場合は, 最後のデータバイトの出力時にEOIがTRUEになります.

マスタモード時において〈リスナアドレス〉が省略された場合は, UNLコマンド, MTAは出力されません. この場合は, すでに1つ以上のリスナとマイトーカがアドレスされていなければなりません.

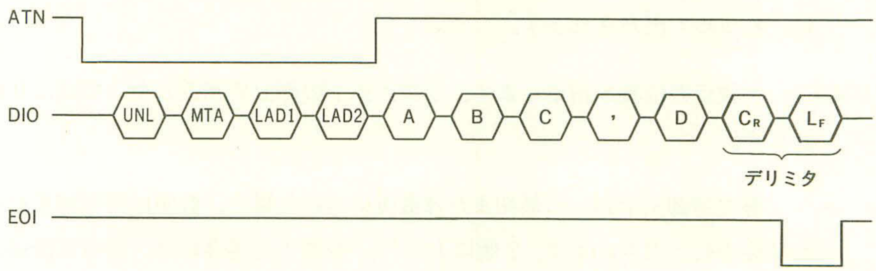
**書式 2)** スレーブモードの場合に使用します.

スレーブモードでは, アドレスの指定ができないため, アドレスは省略しなければな

りません．

この命令が実行されると，コントローラからトーカがアドレスされるまで待ちます．MTA受信後，ATNがfalseになると，データをASCII文字列として送出します．

例）PRINT@1，2；"ABC","D"　@を実行した場合のバスタイミング




PRINT@タイミング

参照：INPUT@，LINE　INPUT@

# PRINT USING　C

| 機　能 | 文字列，数値などのデータを編集し，画面に出力します． |
|---|---|
| 書　式 | PRINT　USING　〈書式制御文字列〉；〈式〉〔｛,｜;｝〈式〉…〕〔｛,｜;｝〕 |
| 文　例 | PRINT　USING　"#####,."；A，B，C |

〈書式制御文字列〉によって決定される領域や書式に従って，〈式〉に指定された各種データを画面に出力します．複数の〈式〉を並べる場合，区切り記号としてはコンマ(,)またはセミコロン(；)のいずれを使っても結果は変りません．

〈書式制御文字列〉中に指定できる書式制御文字は，次のとおりです．

**●文字変数や文字列の書式制御文字**

!……………与えられた文字変数や文字定数の最初の1文字だけを出力します．

&〈n個の□(空白)〉&…与えられた文字変数や文字定数の先頭から(n+2)文字の文字列を出力します．与えられた文字列が(n+2)文字より長い場合は余分な文字は無視され，短い場合には文字列は左づめで出力され，残った部分には空白が出力されます．

@……………文字列全体の出力に使います．複数個指定した場合，1つの〝@″に対して〈式〉の中の1つの文字列が，代入され出力されます．〝@″の数が〈式〉の個数より多い

場合は，余った〝@″は無視されます．

**注意**：日本語文字列を含む文字列を編集出力することはできません．

● **数値の書式制御文字**

# ……………数値を出力する桁数を指定します．指定した桁数より数値の桁数が小さいときには，右づめで出力されます．

．(ピリオド)…小数点の位置を指定します．小数点以下の部分で冗長となる桁には 0 が出力されます．

＋……………〈書式制御文字列〉の最初または最後につけた場合，数値の符号がそれぞれ前または後ろに出力されます．2 個以上の〝＋″を並べた場合には，余分は後述の制御文字以外の文字と同じ扱いとなり，そのまま出力されます．

－(マイナス記号)…〈書式制御文字列〉の最後につけた場合，数値が負の数のときに数値の後ろに〝－″が出力されます．前につけたり，2 個以上並べた場合には後述の制御文字以外の文字と同じ扱いとなり，そのまま出力されます．

＊＊…………〈書式制御文字列〉の先頭につけた場合，数値領域の左側に空白部分ができたとき，そこを〝＊″で埋めて出力します．この〝＊＊″は 2 桁分の領域を確保します．

￥￥…………〈書式制御文字列〉の先頭につけた場合，出力される数値の直前に〝￥″を出力します．〝￥￥″は 2 桁分の領域を確保しますが，このうち 1 桁分は〝￥″の出力領域として使われます．後述の指数形式の書式指定を行った場合には，正しく出力されないことがあります．

＊＊￥………〈書式制御文字列〉の先頭につけた場合，上記の 2 つ(＊＊と￥￥)両方の機能となります．〝＊＊￥″は 3 桁分の領域を確保しますが，このうち 1 桁分は〝￥″の出力領域として使われます．

，(コンマ)…桁数指定の〝#″の並びの中においた場合，数値の整数部が 3 桁毎に〝，″で区切られて出力されます．ただし，上記の〝．″より右側においた場合は，数値の最後に〝，″が出力され，3 桁毎の区切りは行われません．

^^^^…………桁数指定の〝#″の後につけた場合，指数形式(浮動小数点形式)で出力されます．

● **その他の書式制御文字**

＿(下線)…前述の制御文字 1 文字をたんに文字として表示するために使用します．〝＿″に続く 1 文字は常に書式制御機能を持たない文字として出力されます．

● **制御文字以外の文字**

前述の制御文字以外の文字を指定した場合，数値の前や後ろにそのキャラクタが出力されます．ただし，日本語文字は指定できません．

● **数値の領域を超えた場合**

指定した数値領域より数値の桁数が大きい場合，数値の直前に〝%〟が出力されます．数値を丸めた(四捨五入した)ことによって桁数が領域より大きくなった場合も，丸めた数値の前に〝%〟が出力されます．

● **複数の書式制御文字を指定した場合**

〈書式制御文字列〉中に複数の書式制御文字を指定した場合，その制御文字の並びが1つのパターンとなって，指定された複数の〈式〉に対し，順に繰り返しあてはめられます．

たとえば，

```
PRINT USING "@=¥¥### ";"BOOKS", 2500,"TICKETS", 1440
```

とすると，〝@=¥¥### 〟のパターンが以降に指定される式の並びにあてはめられる結果，

```
BOOKS=¥2500  TICKETS=¥1440
```

というように出力されます．

参照：LPRINT USING，PRINT，サンプルプログラム13

# PRINT # USING　C

| 機　能 | 文字列，数値などのデータを編集し，ファイルに出力します． |
|---|---|
| 書　式 | PRINT #〈ファイル番号〉，USING 〈書式制御文字列〉；〈式〉〔 , / ; 〈式〉…〕〔 , / ; 〕 |
| 文　例 | PRINT #2，USING "@ ####"；CASE$，NUMBER |

〈ファイル番号〉には，OPEN によって，そのファイルをオープンしたときに使った番号を指定します．

PRINT# USING は，〈式〉に指定した文字列や数値を〈書式制御文字列〉の形式にもとづいて編集し，ファイルに出力します．

この命令は，その対象がファイルであることを除けば PRINT USING と機能は同じです．

参照：LPRINT # USING，OPEN，PRINT #，PRINT USING

# PSET

Ⓒ

| | |
|---|---|
| 機　能 | 画面上の任意の座標にドットを表示します． |
| 書　式 | PSET (Wx, Wy) / STEP(x, y) 〔,〈パレット番号〉〕 |
| 文　例 | PSET (100, 100), 4 |

ワールド座標(Wx, Wy)の位置にドットを表示します．

(Wx, Wy)の代わりに，STEP をつけて相対座標(x, y)で指定することもできます．

〈パレット番号〉でドットの色を指定します．省略した場合は，現在〔1〕COLOR によって設定されているフォアグラウンドカラーが採用されます．

PSET を実行すると，LP(最終参照点)は，指定した座標点に変更されます．

参照：〔1〕COLOR,〔2〕COLOR, POINT, PRESET, サンプルプログラム 6, 17

# PUT

Ⓒ

| | |
|---|---|
| 機　能 | ファイルバッファ中のデータをファイルに書き出します． |
| 書　式 | PUT 〔#〕〈ファイル番号〉〔,〈数式〉〕 |
| 文　例 | PUT #3, COUNT<br>PUT 2 |

〈ファイル番号〉で指定されたファイルに，対応するバッファの内容を書き出します．

PUT は，指定されたファイルがディスクファイルか，またはプリンタあるいはスクリーンファイルかによってその動作が異なります．

## (1) ディスクファイルの場合

PUT はランダムバッファ中のデータをランダムファイルに書き出します．指定されたファイルはランダムモードでオープンされていなければなりません．

〈数式〉はファイルのレコード番号として解釈され，指定されたレコードにバッファ中のデータが書き出されます．レコード番号の最小値は 1，最大値は 65535 です．〈数式〉が省略された場合には，直前に行われた, GET, PUT で指定されたレコードの次のレコードに書き出します．

(2) プリンタ(LPT：，LPT1：)あるいはスクリーンファイル(SCRN：)の場合

バッファ中のデータを，プリンタあるいは画面に出力します．指定されたファイルは出力モードでオープンされていなければなりません．

〈数式〉は，バッファから，ファイルに対して書き出す文字(バイト)数と解釈されます．0から255までの値で指定します．〈数式〉が省略されたとき，および0が指定されたときには256文字を書き出します．

**注意**：書き出すデータはあらかじめFIELD, LSET／RSETにより準備しておかねばなりません．

**参照**：FILED，GET，LSET／RSET，OPEN，サンプルプログラム1，26，29

# PUT@ C

| | |
|---|---|
| 機　能 | グラフィックパターンや漢字を画面に表示します． |
| 書　式 | 1) PUT〔@〕(Sx, Sy)，〈配列変数名〉〔(〈添字〉)〕〔，〈条件〉〕〔，〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉〕<br>2) PUT〔@〕(Sx, Sy)，KANJI(〈漢字コード〉)〔，〈条件〉〕〔，〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉〕 |
| 文　例 | PUT@　(100, 100)，G%，PSET<br>PUT　(0, 0)，KANJI(&H3021) |

**書式 1)**

GET@によって配列変数に読み込まれたグラフィックパターンを，画面上の指定座標に表示します．

座標(Sx, Sy)はGET@の場合と同様で，ワールド座標でなくスクリーン座標で指定します．

〈配列変数名〉には，表示したいグラフィックパターンが格納されている配列変数の名前を指定します．

〈添字〉は，配列変数内に格納されている複数のグラフィックパターンのうち，どの要素から表示し始めるのかを指定するものです．省略した場合は，配列の最初から表示し始めます．〈添字〉の指定は，GET@でパターンを格納したときの値と対応させるようにしてください．

〈条件〉とは，グラフィックパターンを画面に表示する際のいろいろな条件を指定するもので，以下のものが用意されています．

PSET　：　配列内のグラフィックパターンをそのまま表示します．

PRESET：　白黒モードの場合は配列内のパターンを反転して表示します．カラーモードの場

合は各ドットのパレット番号を，7−(そのドットのパレット番号)(4096色中・16色モードの場合は，15−(そのドットのパレット番号))として表示します．

OR　：　配列内のグラフィックパターンと，すでにある画面上のグラフィックパターンとをドットごとにOR(論理和)し，その結果を画面に表示します．

AND　：　配列内のパターンと画面上のパターンをドットごとにAND(論理積)し，その結果を画面上に表示します．

XOR　：　配列内のパターンと画面上のパターンをドットごとにXOR(排他的論理和)し，その結果を画面に表示します．

〈条件〉を省略した場合はXORが指定されたものとみなされます．

これらの〈条件〉は画面モードによって演算の対象が違います．白黒モードの場合，ドットがあるかないかを対象とし，カラーモードの場合，ドットごとのパレット番号を対象とします．たとえばすでに画面にあるドットの色がパレット番号3で，配列内のグラフィックパターンの色がパレット番号6のときに〝AND〞を指定すると，パレット番号2が表示されます(0011 AND 0110 → 0010)．

〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉は，白黒モードのときにGET@で読み込んだパターンをカラーモードにおいて表示するときのみ，有効なオプションパラメータです．この2つのパラメータは両方とも指定するか，両方とも指定しないかのどちらかしか許されません．

〈フォアグラウンドカラー〉は，白黒モードで読み込んだ際に白であったドットに対しての色指定で，〈バックグラウンドカラー〉は，同様に黒であったドットに対しての色指定です．カラーモードにおいて，それぞれをパレット番号によって指定すると任意の色に変えることができます．

**書式2)**

〈漢字コード〉で指定された漢字または非漢字をグラフィック画面に表示します．

使用できる文字は，JIS第一水準，第二水準および利用者定義文字です．これらの中から任意の文字を〈漢字コード〉(JISコード．各機種本体に添付されているハードウェアマニュアル参照)によって指定することにより，画面上に日本語文字を表示することができます．使い方および機能は，〈配列変数名〉(〈添字〉)の代わりにKANJI(〈漢字コード〉)を使用することを除き，書式1)の場合と同様です．

利用者定義文字は，KPLOADによってシステムに登録することができます．

なお，1つのPUT@では1つの漢字しか表示することはできません．

書式1)，2)とも，この命令を実行するとLP(最終参照点)は(Sx，Sy)に移動します．

**注意**：GET@とPUT@は，〈フォアグラウンドカラー〉，〈バックグラウンドカラー〉を指定し

て，白黒モードで読み込んだパターンをカラーモードで表示する用途の他は，原則として同一画面モードで使用するようにしてください．

参照：GET@，KPLOAD，サンプルプログラム18，19

# RANDOMIZE　C

| | |
|---|---|
| 機　能 | 新しい乱数系列を設定します． |
| 書　式 | RANDOMIZE　〔<式>〕 |
| 文　例 | RANDOMIZE　230 |

<式>に，新しい乱数の種(seed)を指定することによって，RNDで得られる乱数系列を変更します．<式>には数値や変数などを－32768から32767の範囲で指定します．<式>を省略すると，メッセージが表示されてseedを要求してきますので，<式>に同様な範囲の値を入力してください．もし，この範囲外の値を入力すると〝Overflow〞エラーとなります．

参照：RND，サンプルプログラム32

# RBYTE　G　C

| | |
|---|---|
| 機　能 | マルチラインメッセージを送出した後，バイナリデータを受信します． |
| 書　式 | 1）RBYTE　<コマンド>〔，<コマンド>〕；<数値変数>〔，<数値変数>〕 |
| | 2）RBYTE；<数値変数>〔，<数値変数>〕 |
| 文　例 | RBYTE　&H3F，&H41，&H22；A% |

書式 1）マスタモードの場合に使用します．

ATN(アテンション)をtrueにして<コマンド>を送出した後，ATNをfalseにして，バスに出力されてくるバイナリデータを受信し，数値変数に代入します．

<コマンド>は0～255（&H00～&HFF）までのバイナリデータを指定します．

書式 2）スレーブモードの場合に使用します．

この命令が実行されると，コントローラからリスナとしてアドレスされるまで待ちます．

MLA(マイリスンアドレス)を受信後，ATNがfalseになると，バスに出力されてくるバイナリデータを受信して数値変数に代入します．

スレーブモードでは，マルチラインメッセージの送出はできないため，<コマンド>は

すべて省略しなければなりません．

参照：INPUT@，LINE　INPUT@，WBYTE

# READ　C

| | |
|---|---|
| 機　能 | DATA で用意した数値や文字のデータを読み込み，変数に代入します． |
| 書　式 | READ　<変数>〔，<変数>…〕 |
| 文　例 | READ　ZIP，ADDR$ |

READ は DATA と組み合わせて使わなければなりません．READ は DATA 行中に指定されたデータを1対1の対応で変数に割り当てていきます．

<変数>は，DATA 行中で指定したデータが文字定数である場合は文字変数でなくてはなりません．しかし，数値定数の場合は文字変数，数値変数のいずれにも読み込ませることができます．

1つの READ で，複数の DATA 行を順番に参照したり，またいくつかの READ で1つの DATA 行を参照したりすることができます．

READ 中の<変数>の数が DATA のデータ数を超えてしまった場合は，〝Out　of　DATA〟エラーとなります．指定された<変数>が DATA のデータ数よりも少ない場合には，読まれなかったデータから，その次の READ が読み始めます．もしそれ以上 READ がない場合には，余分のデータは無視されます．

始めから，あるいは途中から DATA を読み直すには，RESTORE を使います．

注意：READ では，型が一致しない場合には，〝Type mismatch〟エラーではなく，〝Syntax error〟エラーが起こりますので注意してください．

参照：DATA，RESTORE，サンプルプログラム 3，14

# REM　C

| | |
|---|---|
| 機　能 | プログラムに注釈文を入れます． |
| 書　式 | REM / ' 　〔<注釈文>〕 |
| 文　例 | REM　＊＊＊　Main-Program　＊＊＊ |

REM に続く文字列は，プログラムの注釈になるだけで，プログラムの実行にまったく影響を

与えません．LISTを実行すると入力した内容がそのまま出力されます．REMの代わりにアポストロフィ(')を使うこともできます．

注意：REMに続く注釈文の後に，コロン(：)で区切って他の命令を続けても無視されてしまいます．

コンパイラにおいて，REMは次に示す特殊機能を持ちます．

REMは，一般的には注釈文を指定するためのものですが，次の形式で指定した場合，コンパイラに対するファイル情報の宣言文として機能します．

REM　$FILE　〔<ファイル同時オープン数>〕〔，<レコードサイズ>〕

〈ファイル同時オープン数〉には，同時オープンするファイルの個数を指定します(インタプリタの場合，この値は起動時のスイッチ(／F)で指定します)．省略された場合，2が設定されます．

〈レコードサイズ〉には，ディスクファイルアクセスのためのレコードサイズを指定します．ここで指定したサイズのバッファが〈ファイル同時オープン数〉で指定した個数分確保されます．この値はFIELDでランダムファイルバッファに変数を割り当てる際の，各フィールド長の和の制限値となります(インタプリタの場合，この値は起動時のスイッチ(／S)で指定します)．省略された場合，256が設定されます．

# RENUM

| | |
|---|---|
| 機　能 | プログラムの行番号を新しくつけ直します． |
| 書　式 | RENUM　〔<新行番号>〕〔，<旧行番号>〕〔，<増分>〕 |
| 文　例 | RENUM　1000，10 |

〈新行番号〉は，新しくつける行番号の最初の行番号で，省略すると10が採用されます．

〈旧行番号〉は行番号のつけ替えを始める現在のプログラムの行番号です．省略するとそのプログラムの最初の行番号が採用されます．

〈増分〉は新しくつける各行番号のあいだの増分で，省略すると10となります．

RENUMは，GOTO，GOSUB，THEN，ELSE，ON…GOTO，ON…GOSUBおよびERLで参照している行番号も新しい行番号に対応して変更します．もし，これらの命令の参照する行番号が存在しない場合には，次のようなエラーメッセージが表示されます．

"Undefinde　line　xxxx　in　yyyy"

これは，行番号yyyy中に，参照されていない行番号xxxxがある，という意味です．この場合，参照されていない行番号xxxxはRENUMによって変更されませんが，行番号yyyyは変更されてしまいますのでエラーの発生した行番号の所在が分からなくなってしまいます．したがって，RENUMを行う前にプログラムをセーブしておいたほうが安全です．

コンパイラにおいてはこの機能は使用できません．

注意：65529を超える行番号は発生することができません．このような場合には"Illegal function call"エラーとなります．

# RESTORE

C

機　能　READで読むDATA行の先頭行を指定します．

書　式　RESTORE　〔〈行番号〉〕

文　例　RESTORE

RESTORE　800

〈行番号〉を指定すると，READはその行番号のDATA行からデータを読み始めます．省略した場合はプログラム中の最初のDATA行から読み始めます．

参照：DATA，READ，サンプルプログラム3

# RESUME

C

機　能　エラー処理ルーチンを終了し，元のプログラムの実行を再開します．

書　式　1）RESUME　〔0〕

2）RESUME　NEXT

3）RESUME　〈行番号〉

文　例　RESUME　＊START

書式1）エラーの原因となった文からプログラムの実行を再開します．この場合0は省略できます．

書式2）エラーの原因となった文の次の文からプログラムの実行を再開します．

書式3）〈行番号〉で指定した行から実行を再開します．

参照：ON　ERROR　GOTO，サンプルプログラム22

# RETURN　C

機　能　サブルーチンまたは割り込み処理ルーチンを終了し，元のプログラムの実行を再開します.

書　式　RETURN〔<行番号>〕

文　例　RETURN

RETURN　200

サブルーチンの実行を終了し，そのサブルーチンを呼び出したGOSUBの次の命令から実行を再開します.RETURNは割り込み処理ルーチンの終了にも使いますが，この場合RETURNは，割り込みの生じた命令の次の命令から実行を再開します．1つのサブルーチン内に複数のRETURNがあってもかまいません.

RETURNでは，<行番号>を指定して，特定の行にリターンさせることもできます．ただし，サブルーチンの多重化を行っている場合や，FOR～NEXTの内部からGOSUBでサブルーチンを呼び出している場合，リターン先の<行番号>には，そのサブルーチンの呼び出し側と同じスタックレベルのプログラム行を指定しなければなりません．これを無視すると，スタック領域が異常消費され，プログラムは正しく動作しません．とくに必要がない限り，使用しないようにしてください.

注意：RETURNを単独で使うことはできません．プログラムの実行中，GOSUBなしでRETURNに出合うと，"RETURN　without　GOSUB" エラーが起こります.

参照：GOSUB, ON COM GOSUB, ON…GOSUB／ON…GOTO, ON HELP GOSUB, ON KEY GOSUB, ON STOP GOSUB, ON TIME$ GOSUB, サンプルプログラム7

# RIGHT$　関 数　C

機　能　文字列の右側から任意の長さの文字列を抜き出します.

書　式　RIGHT$(<文字列>，<式>)

文　例　PRINT　RIGHT$("ABCDabcd", N*4)

<文字列>の右側(最後)から<式>で指定した桁数の文字列を抜き出します．<式>の値は0から255の範囲になければなりません.

<式>の値が0ならば，RIGHT$の値はヌルストリング(空の文字列)となります．<式>が<文字列>の文字数より大きければ，RIGHT$の値は<文字列>とまったく同じになります.

参照：LEFT$, MID$, サンプルプログラム25

# RMDIR　C

| 機能 | ディスクのディレクトリを削除します. |
| --- | --- |
| 書式 | RMDIR 〔〈ドライブ名〉〕〈ディレクトリ名〉 |
| 文例 | RMDIR "BIN"<br>RMDIR "B:¥SOURCE"<br>RMDIR "A:BIN¥SUB"<br>RMDIR "..¥TEST" |

〈ドライブ名〉で指定されたドライブにあるディスクの,〈ディレクトリ名〉で指定されるディレクトリを削除します.

〈ドライブ名〉が省略された場合は,カレントドライブにあるディスクが指定されたものとみなされます.

〈ディレクトリ名〉には,削除したいディレクトリの名前を,¥(ルート)で始まる絶対指定か,現在のカレントディレクトリからの相対指定によって指定します.

指定したディレクトリにファイルが登録されている場合は,ディレクトリの削除を行うことはできません.このような場合は,登録されているファイルをすべて削除(KILL を使用)した後,RMDIR を実行します.

例) RMDIR "B:¥SOURCE"

Bドライブにあるディスクのルートディレクトリ内のSOURCEという名のディレクトリを削除(絶対指定)します.

RMDIR "BIN"

カレントドライブにあるディスクのカレントディレクトリ内のBINという名のディレクトリを削除(相対指定)します.

ディレクトリが階層化されている場合には,複数の〈ディレクトリ名〉を¥でつないで指定してください.

例) RMDIR "A:¥BIN¥SUB"

Aドライブにあるディスクのルートディレクトリのさらに下にあるBINディレクトリ内の,SUBという名のディレクトリを削除します.

なお,カレントディレクトリのすぐ上にあるディレクトリを,".."(ピリオド2個)で表すことができますので,相対指定を行う場合に利用することができます.

例）　RMDIR　".. ¥TEST"

カレントドライブにあるディスクのカレントディレクトリのすぐ上にあるディレクトリ内の TEST という名のディレクトリを削除します．

注意：カレントディレクトリ自身を削除することはできません．また，カレントディレクトリを絶対指定で表したときに途中に表れるディレクトリは，すべて削除することができません．たとえば，カレントディレクトリが¥BIN¥BASIC のときに，

```
RMDIR  "BASIC"
RMDIR  ".."
RMDIR  "¥BIN"
```

などは，いずれも実行することができません．

参照：CHDIR，KILL，MKDIR

## RND 関数 C

機能　乱数を得ます．

書式　RND〔(<数式>)〕

文例　R=RND

R=RND(−3)

0 以上 1 未満の乱数を得ます．得られる乱数は，RUN および CLEAR が実行されるごとにいつも同系列となります．ただし，RANDOMIZE を使用すれば，この系列を変えることが可能です．

RND 関数の機能は，指定する〈数式〉の値によって次のように異なります．

| | | |
|---|---|---|
| 負の数 | ： | 乱数系列を初期化する． |
| 0 | ： | 1 つ前に発生した乱数の値をとる(繰り返す)． |
| 正の数 | ： | 次の乱数を発生する． |

なお，(<数式>)を省略して(カッコも含む)，たんに "RND" とした場合は，正の数を指定したのと同じ機能になります．

参照：RANDOMIZE，サンプルプログラム32

# ROLL

C

機　能　グラフィック画面を上下あるいは左右にスクロールさせます．

書　式　ROLL 〔<上下方向ドット数>〕〔，<左右方向ドット数>〕〔，｜N / Y｜〕

文　例　ROLL　−16，32，Y

ROLL　，4

<上下方向ドット数> には，画面の縦方向のドットの数を指定します．許される値の範囲は640×200ドットの画面モードでは−199から199，640×400ドットの画面モードでは−399から399までの値で，正の場合には上方向に，負の場合には下方向に，指定のドット数(絶対値)だけスクロールします．省略した場合は，縦方向のスクロールは行いません．

<左右方向ドット数> には，画面の横方向のドットの数を指定します．許される値の範囲は-639から639までの値で，正の場合には左方向に，負の場合には右方向に，指定のドット数(絶対値)以下で最も大きい8の倍数と等しいドット数だけ，スクロールします．省略した場合は，左右のスクロールは行いません．

NまたはYを指定すると，スクロールにより新たに現れた領域を決められた色でクリアすることができます．Yを指定するとバックグラウンドカラーでクリアし，Nを指定するか省略したときには，パレット番号0でクリアします．

参照：サンプルプログラム19

# RUN

C

機　能　メモリにあるプログラムの実行を開始します．また，ディスクからプログラムをメモリにロードし，そのプログラムを実行します．

書　式　1) RUN 〔<行番号>〕

2) RUN 〈ファイルディスクリプタ〉〔，R〕

文　例　RUN　＊MAIN

RUN　"B : TEST"

RUNは，プログラムの実行に先だち，変数をすべて初期化し，オープンされているすべてのデータファイルを閉じます．

**書式1)** <行番号>を指定すると，メモリ上のプログラムの，その行から実行が始まります．省略すると，最も若い行番号の行から実行が始まります．プログラムの実行が終るとコマ

ンドレベルにもどります.

コンパイラによってコンパイルされたプログラム内のRUNの場合,プログラムの実行が終了すると,MS-DOSに制御がもどります.

書式2)〈ファイルディスクリプタ〉で指定されたディスク上のプログラムをロードし,ただちに実行します.したがって,メモリ上にプログラムがあった場合,そのプログラムは消去されてしまいますので注意してください.プログラムの実行が終るとコマンドレベルにもどります.

Rオプションをつけた場合には,プログラムの実行前にデータファイルは閉じられず,オープンされたままでプログラムが実行されます.

コンパイラによってコンパイルされたプログラムの場合,R指定はできません.ファイルは無条件に閉じられた後,指定されたプログラムのロード実行が行われます.

また起動するプログラムのファイル名にディレクトリ名を指定することはできません.CHILDと同様,カレントディレクトリ下のファイルのみが処理の対象となります.特定のディレクトリ下のプログラムを処理対象としたい場合,ENVIRONを使用し,環境文字列テーブルにディレクトリ名を設定しておく必要があります.起動するプログラムは他の言語で作成されたEXE形式あるいはCOM形式のプログラムであってもかまいません.

RUNはCHAINとちがい,起動したプログラムに制御が渡った後も,呼び出したプログラムの一部の情報がメモリ上に残されます.したがって,次々にRUNが実行される場合,〝Out of memory〞のエラーが発生することがあります.このような場合,CHAINを使用してください.

参照:CHAIN, CHILD, ENVIRON, GOTO/GO TO, LOAD

# SAVE

| 機能 | メモリにあるBASICプログラムをファイルにセーブします. |
|---|---|
| 書式 | SAVE 〈ファイルディスクリプタ〉〔, {A｜P}〕 |
| 文例 | SAVE "B:MYPROG", A |

〈ファイルディスクリプタ〉で指定されるファイル(ディスク,RS-232C回線)に,メモリ上のプログラムをセーブ(書き出し)します.指定したファイルディスクリプタと同じ名前のファイ

ルが存在した場合には，古い内容は失われ，新しいものに更新されます．

Aオプションが指定された場合には，プログラムはアスキー形式でセーブされます．つまり，プログラムをLISTで表示したときと同じイメージでセーブが行われます．

Aオプションの指定がない場合は，プログラムはバイナリ形式に圧縮されてセーブされます．つまり，メモリに格納されているときのイメージでそのままセーブされます．

Pオプションが指定された場合には，プログラムは暗号化されたバイナリ形式でセーブされます．Pオプションは，セーブされたプログラムを，読み出し，書き込み，変更といった操作から保護するための機能を持っています．後でメモリにロードしても，LISTやEDITにより内容を見たり，変更しようとすると，“Illegal　function　call”エラーとなります．

コンパイラにおいてはこの機能は使用できません．

注意：アスキー形式のセーブは，バイナリ形式よりも，多くのファイルスペースを必要としますが，コマンドによってはプログラムがアスキー形式でセーブされていることが条件となることがあります．たとえば，MERGEコマンドは，アスキー形式のファイルを必要とします．また，アスキー形式でセーブされたファイルは，データファイルとして読み出すことができます．

Pオプションによって一度プログラムが暗号化されると，これを解除する方法は用意されておらず，二度と内容の変更などはできなくなるので注意してください．

参照：BLOAD，BSAVE，LOAD，MERGE

# SCREEN　C

**機　能**　グラフィック画面に対して種々のモードを設定します．

**書　式**　SCREEN〔<画面モード>〕〔，<画面スイッチ>〕〔，<アクティブページ>〕〔，<ディスプレイページ>〕

**文　例**　SCREEN　1，0，0，7

<画面モード>は，カラー，白黒，分解能など，グラフィック画面の最も基本的なモードを設定するパラメータです．

| 指定値 | | 分解能(横×縦) | 使用可能ページ数 |
|---|---|---|---|
| 0 | カラーモード | 640×200 | 4 |
| 1 | 白黒モード | 640×200 | 12(16＊) |
| 2 | 高分解能白黒モード | 640×400 | 6(8＊) |
| 3 | 高分解能カラーモード | 640×400 | 2 |

ここで，＊のついた数値は，16色モードのときの値を表します.

**注意**：機種によって各モードともページ数が半減します．また，16色モードは機種の違い，あるいは16色グラフィックボードの有無により，使用できない場合があります．各機種本体に添付されているユーザーズマニュアルを参照してください．

〈画面スイッチ〉は，指定する値により，現在グラフィック画面に描かれているパターンなどを一時的に消去するものです．

| 指定値 | |
|---|---|
| 0（または1） | グラフィック画面の表示を行う． |
| 2（または3） | 現在グラフィック画面に表示されている図形などを一時的に消去する． |

〈アクティブページ〉と〈ディスプレイページ〉はグラフィック命令で書き込むページと表示するページの選択を行うためのものです．

〈アクティブページ〉と〈ディスプレイページ〉に指定できる値とその意味は8色モード(パレットモードが8色中・8色モードの場合あるいは4096色中・8色モードの場合)と16色モード(パレットモードが4096色中・16色モードの場合)で異なります(パレットモードに関しては〔1〕COLORを参照してください)．

### (1) 8色中･8色モードあるいは4096色中･8色モードの場合

〈アクティブページ〉にはグラフィック命令によって書き込むページを指定します．画面モードによって扱える画面数は異なります．

〈アクティブページ〉に指定できる値とページ番号との対応関係

| 指定値 | 画面モード | 書き込まれるページ番号 |
|---|---|---|
| 0～3 | カラーモード(0) | 1～4 |
| 0～11 | 白黒モード(1) | 1～12 |
| 0～5 | 高分解能白黒モード(2) | 1～6 |
| 0，1 | 高分解能カラーモード(3) | 1，2 |

**注意**：機種によって各モードともページ数が半減します．その場合，〈アクティブページ〉に指定できる値もここに記載した指定値の1/2に制約されますのでご注意ください．たとえば，画面モードが0の場合0から1まで，また画面モードが3の場合0のみとなります．各機種本体に添付されているユーザーズマニュアルを参照してください．

〈ディスプレイページ〉には，画面モードに対応したページのうちどのページを表示するかを指定します．

〈ディスプレイページ〉に指定できる値と表示されるページ番号との対応関係

| 指定値 | カラーモード (0) | 白黒モード (1) | 高分解能白黒モード (2) | 高分解能カラーモード (3) |
|---|---|---|---|---|
| 0 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 1 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 |
| 2 | ページ2のみ表示 | ページ2のみ表示 | ページ2のみ表示 | × |
| 3 | × | ページ1，2を合成表示 | ページ1，2を合成表示 | × |
| 4 | × | ページ3のみ表示 | ページ3のみ表示 | × |
| 5 | × | ページ1，3を合成表示 | ページ1，3を合成表示 | × |
| 6 | × | ページ2，3を合成表示 | ページ2，3を合成表示 | × |
| 7 | × | ページ1，2，3を合成表示 | ページ1，2，3を合成表示 | × |
| 8 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 9 | × | ページ4のみ表示 | × | × |
| 10 | × | ページ4のみ表示 | × | × |
| 11 | × | ページ4，5を合成表示 | × | × |
| 12 | × | ページ6のみ表示 | × | × |
| 13 | × | ページ4，6を合成表示 | × | × |
| 14 | × | ページ5，6を合成表示 | × | × |
| 15 | × | ページ4，5，6を合成表示 | × | × |

**注意**：機種によって各モードともページ数が半減します．その場合〈ディスプレイページ〉に指定できる値はここまでです．各機種本体に添付されているユーザーズマニュアルを参照してください．

| 指定値 | カラーモード (0) | 白黒モード (1) | 高分解能白黒モード (2) | 高分解能カラーモード (3) |
|---|---|---|---|---|
| 16 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 17 | ページ3のみ表示 | ページ7のみ表示 | ページ4のみ表示 | ページ2のみ表示 |
| 18 | ページ4のみ表示 | ページ8のみ表示 | ページ5のみ表示 | × |
| 19 | × | ページ7，8を合成表示 | ページ4，5を合成表示 | × |
| 20 | × | ページ9のみ表示 | ページ6のみ表示 | × |
| 21 | × | ページ7，9を合成表示 | ページ4，6を合成表示 | × |
| 22 | × | ページ8,9を合成表示 | ページ5，6を合成表示 | × |
| 23 | × | ページ7，8，9を合成表示 | ページ4，5，6を合成表示 | × |
| 24 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 25 | × | ページ10のみ表示 | × | × |
| 26 | × | ページ11のみ表示 | × | × |
| 27 | × | ページ10,11を合成表示 | × | × |
| 28 | × | ページ12のみ表示 | × | × |
| 29 | × | ページ10,11を合成表示 | × | × |
| 30 | × | ページ11,12を合成表示 | × | × |
| 31 | × | ページ10,11,12を合成表示 | × | × |

×印：指定不可

(2) 4096色中・16色モードの場合

〈アクティブページ〉にはグラフィック命令によって書き込むページを指定します．画面モードによって扱える画面数は異なります．

〈アクティブページ〉に指定できる値とページ番号との対応関係

| 指定値 | 画面モード | 書き込まれるページ番号 |
|---|---|---|
| 0～3 | カラーモード(0) | 1～4 |
| 0～15 | 白黒モード(1) | 1～16 |
| 0～7 | 高分解能白黒モード(2) | 1～8 |
| 0，1 | 高分解能カラーモード(3) | 1，2 |

**注意**：機種によって各モードともページ数が半減します．その場合〈アクティブページ〉に指定できる値もここに記載した指定値の1/2に制約されますのでご注意ください．たとえば，画面モードが0の場合1まで，画面モードが2の場合3まで，また画面モードが3の場合0のみとなります．各機種本体に添付されているユーザーズマニュアルを参照してください．

〈ディスプレイページ〉には，画面モードに対応したページのうちどのページを表示するかを指定します．

〈ディスプレイページ〉に指定できる値と表示されるページ番号との対応関係

| 指定値 | カラーモード(0) | 白黒モード(1) | 高分解能白黒モード(2) | 高分解能カラーモード(3) |
|---|---|---|---|---|
| 0 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 1 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 | ページ1のみ表示 |
| 2 | ページ2のみ表示 | ページ2のみ表示 | ページ2のみ表示 | × |
| 3 | × | ページ1，2を合成表示 | ページ1，2を合成表示 | × |
| 4 | × | ページ3のみ表示 | ページ3のみ表示 | × |
| 5 | × | ページ1，3を合成表示 | ページ1，3を合成表示 | × |
| 6 | × | ページ2，3を合成表示 | ページ2，3を合成表示 | × |
| 7 | × | ページ1，2，3を合成表示 | ページ1，2，3を合成表示 | × |
| 8 | × | ページ4のみ表示 | ページ4のみ表示 | × |
| 9 | × | ページ1，4を合成表示 | ページ1，4を合成表示 | × |
| 10 | × | ページ2，4を合成表示 | ページ2，4を合成表示 | × |
| 11 | × | ページ1，2，4を合成表示 | ページ1，2，4を合成表示 | × |
| 12 | × | ページ3，4を合成表示 | ページ3，4を合成表示 | × |
| 13 | × | ページ1，3，4を合成表示 | ページ1，3，4を合成表示 | × |
| 14 | × | ページ2，3，4を合成表示 | ページ2，3，4を合成表示 | × |
| 15 | × | ページ1,2,3,4を合成表示 | ページ1,2,3,4を合成表示 | × |
| 16 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 17 | × | ページ5のみ表示 | × | × |
| 18 | × | ページ6のみ表示 | × | × |
| 19 | × | ページ5，6を合成表示 | × | × |
| 20 | × | ページ7のみ表示 | × | × |
| 21 | × | ページ5，7を合成表示 | × | × |
| 22 | × | ページ6，7を合成表示 | × | × |
| 23 | × | ページ5，6，7を合成表示 | × | × |
| 24 | × | ページ8のみ表示 | × | × |
| 25 | × | ページ5，8を合成表示 | × | × |
| 26 | × | ページ6，8を合成表示 | × | × |
| 27 | × | ページ5，6，8を合成表示 | × | × |

| 指定値 | カラーモード (0) | 白黒モード (1) | 高分解能白黒モード (2) | 高分解能カラーモード (3) |
|---|---|---|---|---|
| 28 | × | ページ7, 8を合成表示 | × | × |
| 29 | × | ページ5, 7, 8を合成表示 | × | × |
| 30 | × | ページ6, 7, 8を合成表示 | × | × |
| 31 | × | ページ5, 6, 7, 8を合成表示 | × | × |

注意：機種によって各モードともページ数が半減します．その場合〈ディスプレイページ〉に指定できる値はここまでです．各機種本体に添付されているユーザーズマニュアルを参照してください．

| 指定値 | カラーモード (0) | 白黒モード (1) | 高分解能白黒モード (2) | 高分解能カラーモード (3) |
|---|---|---|---|---|
| 32 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 33 | ページ3のみ表示 | ページ9のみ表示 | ページ5のみ表示 | ページ2のみ表示 |
| 34 | ページ4のみ表示 | ページ10のみ表示 | ページ6のみ表示 | × |
| 35 | × | ページ9, 10を合成表示 | ページ5, 6を合成表示 | × |
| 36 | × | ページ11のみ表示 | ページ7のみ表示 | × |
| 37 | × | ページ9, 11を合成表示 | ページ5, 7を合成表示 | × |
| 38 | × | ページ10, 11を合成表示 | ページ6, 7を合成表示 | × |
| 39 | × | ページ9, 10, 11を合成表示 | ページ5, 6, 7を合成表示 | × |
| 40 | × | ページ12のみ表示 | ページ8のみ表示 | × |
| 41 | × | ページ9, 12を合成表示 | ページ5, 8を合成表示 | × |
| 42 | × | ページ10, 12を合成表示 | ページ6, 8を合成表示 | × |
| 43 | × | ページ9, 10, 12を合成表示 | ページ5, 6, 8を合成表示 | × |
| 44 | × | ページ11, 12を合成表示 | ページ7, 8を合成表示 | × |
| 45 | × | ページ9, 11, 12を合成表示 | ページ5, 7, 8を合成表示 | × |
| 46 | × | ページ10, 11, 12を合成表示 | ページ6, 7, 8を合成表示 | × |
| 47 | × | ページ9, 10, 11, 12を合成表示 | ページ5, 6, 7, 8を合成表示 | × |
| 48 | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない | 全ページ表示しない |
| 49 | × | ページ13のみ表示 | × | × |
| 50 | × | ページ14のみ表示 | × | × |
| 51 | × | ページ13, 14を合成表示 | × | × |
| 52 | × | ページ15のみ表示 | × | × |
| 53 | × | ページ13, 15を合成表示 | × | × |
| 54 | × | ページ14, 15を合成表示 | × | × |
| 55 | × | ページ13, 14, 15を合成表示 | × | × |
| 56 | × | ページ16のみ表示 | × | × |
| 57 | × | ページ13, 16を合成表示 | × | × |
| 58 | × | ページ14, 16を合成表示 | × | × |
| 59 | × | ページ13, 14, 16を合成表示 | × | × |
| 60 | × | ページ15, 16を合成表示 | × | × |
| 61 | × | ページ13, 15, 16を合成表示 | × | × |
| 62 | × | ページ14, 15, 16を合成表示 | × | × |
| 63 | × | ページ13, 14, 15, 16を合成表示 | × | × |

この〈アクティブページ〉と〈ディスプレイページ〉については，8色モードと16色モードでは，指定できる値とその意味が異なることに注意してください．

SCREENを実行すると，そのとき設定されていたウィンドウ，ビューポート，LP(最終参照点)は初期状態にもどります．なお，LPの初期状態はワールド座標，スクリーン座標とも(0, 0)です．

参照：〔1〕COLOR，サンプルプログラム20，21

## SEARCH 関数 C

| | |
|---|---|
| 機　能 | 配列変数の中から指定された値を捜し出し，その要素の順位を得ます． |
| 書　式 | SEARCH(〈配列変数名〉，〈整数表記〉〔，〈開始添字〉〕〔，〈ステップ値〉〕) |
| 文　例 | NUM＝SEARCH(A%，100，0，3) |

〈配列変数名〉で指定された配列変数の要素の中から〈整数表記〉で指定された値を捜し，最初に見つかった要素の添字を得ます．指定された値がその配列変数内で見つからなかった場合，SEARCH の値は－1となります．

〈配列変数名〉は整数型の一次元配列でなければなりません．またこの配列変数は SEARCH の実行に先だって DIM によって宣言されていなければなりません．

〈整数表記〉には捜したい値を指定します．この値は整数でなければならず，実数を指定した場合は整数化してから実行します．

〈開始添字〉には，配列中のどの要素から捜し始めるかを指定します．省略した場合には，配列の最初から捜し始めます．

〈ステップ値〉を指定した場合は，その間隔で捜します．省略した場合は1が採用されます．

SEARCH はランダムアクセスファイルのインデックスを捜し出したり，GET@，PUT@で用いる配列データを捜し出したりするのに使うことができます．

参照：DIM，OPTION　BASE

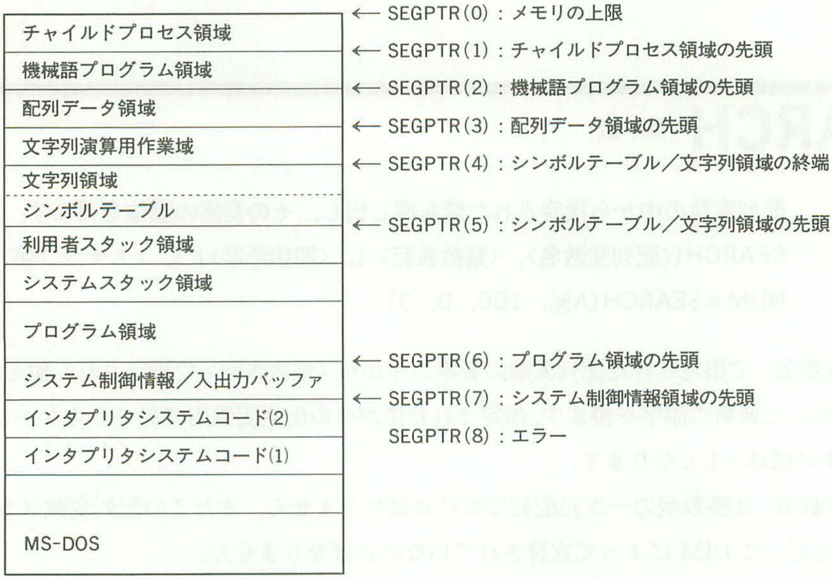
## SEGPTR 関数 C

| | |
|---|---|
| 機　能 | 各種の制御領域のセグメントベースを得ます． |
| 書　式 | SEGPTR (〈機能〉) |
| 文　例 | DEF　SEG＝SEGPTR (2) |

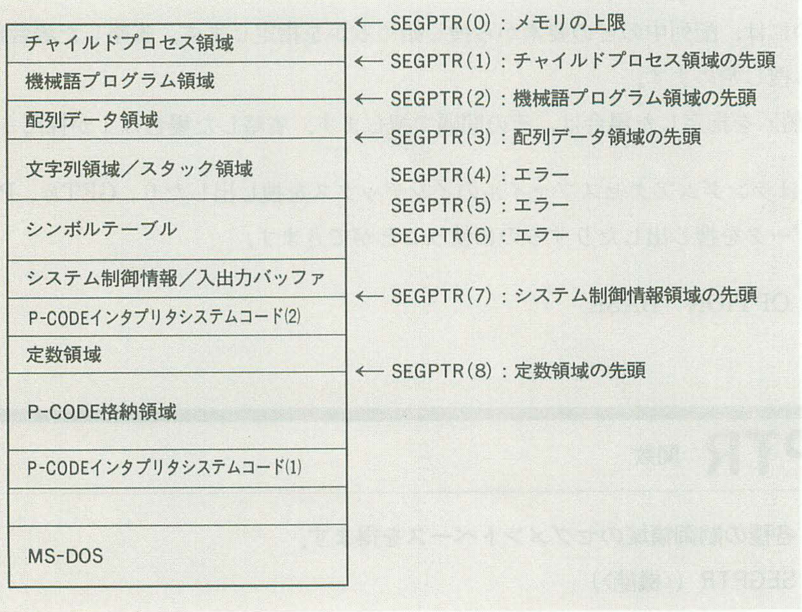
各種の制御領域のセグメントベース（実際の物理アドレスを16で割った値）を得ます．

〈機能〉に指定する値と得られるセグメントベースは，インタプリタとコンパイラにより次のように異なります．

インタプリタ




コンパイルされたプログラムの実行時




参照：CLEAR，DEF　SEG，FRE，VARPTR

# SET　Ⓒ

機能　ディスクファイルに対し書き込み禁止属性のセット・リセットを行います．

書式　1) SET　<ファイルディスクリプタ>, <属性文字>

　　　2) SET　<ファイル番号>, <属性文字>

文例　SET　"B : DATA1", "P"

　　　SET　"B : DATA1", ""

　　　SET　#1, "P"

指定されたファイルに書き込み禁止属性を与えたり，書き込み禁止属性が与えられているファイルの属性を解除したりします．

<属性文字> に P が指定されると，書き込み禁止属性を与えます．

<属性文字>に空白(" ")あるいは空の文字列("")が指定されると，書き込み禁止属性を解除します．

書式 1)

<ファイルディスクリプタ> が指定された場合，指定されたファイルが処理対象となります．書き込み禁止属性は指定されたファイルそのものに与えられます．

書式 2)

<ファイル番号>が指定された場合，対応するファイルが開かれている間だけ，書き込み禁止属性が与えられます．

参照：ATTR$

# SGN　関数　Ⓒ

機能　符号を調べます．

書式　SGN(<数式>)

文例　PRINT　SGN(RESULT)

<数式> に指定された式の値の正負により，SGN の値は次のようになります．

| | | |
|---|---|---|
| 正の場合 | : | 1 |
| 0 | : | 0 |
| 負の場合 | : | −1 |

参照：サンプルプログラム 9

## SIN 関数 C

| | |
|---|---|
| 機　能 | 正弦(サイン)を得ます. |
| 書　式 | SIN(<数式>) |
| 文　例 | X=RADIUS＊SIN(ANGLE)<br>PRINT　SIN(−3.14159/2) |

<数式> の値に対する正弦値を得ます. <数式> の単位はラジアンで指定します.

<数式>に倍精度実数が含まれる場合, 得られる値は倍精度となりますが, 他の場合には単精度となります.

参照:ATN, COS, TAN, サンプルプログラム 6, 30

## SPACE$ 関数 C

| | |
|---|---|
| 機　能 | 任意の長さの空白文字列を得ます. |
| 書　式 | SPACE$(<数式>) |
| 文　例 | QTTY$=NUM$+SPACE$(10)+RES$ |

<数式>に指定した数だけの空白(スペース. キャラクタコード&H20)が連なった文字列を得ます. <数式> の値は 0 から 255 までの範囲の数値でなければなりません.

参照:SPC, サンプルプログラム25

## SPC 関数 C

| | |
|---|---|
| 機　能 | 任意の数だけの空白を出力します. |
| 書　式 | SPC(<数式>) |
| 文　例 | PRINT　"SUPER" ; SPC(10) ; "BASIC" |

出力の対象となる行の中で, <数式> に指定した数だけの空白(スペース)を出力します.

ただし, SPC は, PRINT や LPRINT などの出力文中でのみ使用することができるもので, 文字式としては使えません.

<数式>には, −32768 から 32767 までの範囲の数値を指定できますが, 負の数はすべて 0 とみなされます. また, 正の数の場合でも, WIDTH で決められている現在の水平表示桁数以上の場合は, <数式> をその表示桁数で割った余りを値とします.

参照:SPACE$, TAB

## SQR 関数 C

| | |
|---|---|
| 機　能 | 平方根を得ます. |
| 書　式 | SQR(〈数式〉) |
| 文　例 | DISTANCE＝SQR(X＊X＋Y＊Y) |

〈数式〉に指定した値の平方根(スクエアルート)を得ます.

〈数式〉に倍精度実数が含まれる場合，得られる値は倍精度となりますが，他の場合には単精度となります.

参照：サンプルプログラム8

## SRQ ON／OFF／STOP G C

| | |
|---|---|
| 機　能 | SRQ（サービスリクエスト）の受信による割り込みの許可，禁止，停止を制御します. |
| 書　式 | 1）SRQ　ON<br>2）SRQ　OFF<br>3）SRQ　STOP |
| 文　例 | SRQ　ON |

これらの命令は，マスタモードでのみ使用できます.

書式 1）SRQ の受信による割り込みを，この命令の次の命令を実行した後，許可します．その後，SRQ が受信されるごとに割り込みが発生し，ON　SRQ　GOSUB によって定義された処理ルーチンに分岐します.

書式 2）SRQ の受信による割り込みを禁止します．この命令が実行されると，SRQ を受信しても割り込みは起こらず処理ルーチンには分岐しません.

書式 3）SRQ の受信による割り込みを一時停止します．この命令が実行されると，SRQ を受信しても割り込みは起こらず，処理ルーチンには分岐しません．しかし，割り込みのあったことを覚えているため，後で SRQ　ON によって割り込みが許可されると，前に SRQ の受信があったことによって処理ルーチンに分岐します.

参照：ON　SRQ　GOSUB

# STATUS 関数

G C

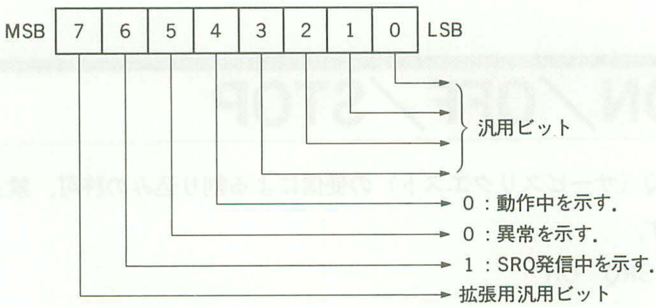
機能　デバイスステータスを格納するための特殊変数です.

書式　STATUS

文例　PRINT　HEX$ (STATUS)

スレーブモード時においてシリアルポールが行われたときに，コントローラにたいして送出するためのデバイスステータスを格納します.

ここに格納されるデバイスステータスは，コントローラよりシリアルポールが行われたときに自動的にバスに送出されます.




ビット6 (SRQ ビット) は，ISET　SRQ によりセットされ，コントローラよりシリアルポールを受けるとリセットされます.

参照：ISET　SRQ

# STATUS DIAL 関数

P

機能　電話機に記憶されている電話番号の機能情報を調べます.

書式　STATUS　DIAL([[#] 〈電話機番号〉,] 〈短縮番号〉)

文例　STATUS　DIAL(#1, 21)

〈電話機番号〉で指定された電話機が記憶している〈短縮番号〉に対応する，自動発信後のモードを得ます.

〈電話機番号〉には，CMD　LINE　OPEN で接続された電話機の番号を指定します.〈電話機番号〉が省略された場合は，1が指定されたものとみなされます.

〈短縮番号〉は1〜40までの数値でなければなりません．得られる値と自動発信後のモードの関係は次のとおりです．

0 ： ダイアル後通話モードとなる
1 ： アンサートーン検出後データ通信モードとなる
2 ： 接続後データ通信モードとなる
3 ： ダイアル後データ通信モードとなる

各値の意味の詳細については，CMD　DIALを参照してください．〈短縮番号〉に対応する電話番号が記憶されていない場合には，−1が得られます．

注意：この関数はオフラインコマンドモードでのみ使用可能です．PC-9863N，PC-9865モデムボードでは，この関数を使用することはできません．

## STATUS DIAL$　関数　P

| | |
|---|---|
| 機　能 | 電話機に記憶されている電話番号を調べます． |
| 書　式 | STATUS　DIAL$(〔〔#〕〈電話機番号〉,〕〈短縮番号〉) |
| 文　例 | STATUS　DIAL$(#1，21) |

〈電話機番号〉で指定されている電話機に記憶されている〈短縮番号〉に対応する，電話番号を得ます．

〈電話機番号〉には，CMD　LINE　OPENで接続された電話機の番号を指定します．〈電話機番号〉が省略された場合は，1が指定されたものとみなされます．

〈短縮番号〉は1〜40の数値でなければなりません．〈短縮番号〉に対応する電話番号が記憶されていない場合にはヌルストリング（空の文字列）が得られます．

注意：この関数はオフラインコマンドモードでのみ使用可能です．PC-9863N，PC-9865モデムボードでは，この関数を使用することはできません．

## STATUS DSKF　関数　N C

| | |
|---|---|
| 機　能 | ドライブに接続されているボリュームの属性を得ます． |
| 書　式 | STATUS　DSKF(〈ドライブ名〉) |
| 文　例 | PRINT　STATUS　DSKF("A：") |

〈ドライブ名〉によって指定されたドライブに現在接続されているボリュームの属性を得ま

す．得られる値とボリュームの属性の対応は次のとおりです．

1　：　LOCAL
2　：　WORK
3　：　PUBLIC
4　：　SYSTEM
0　：　ボリュームが接続されていない

注意：DSKFとは別の関数です．ディスクの残り容量などに関する情報を得る場合は，DSKFを使ってください．

参照：DSKF，STATUS　DSKI$

## STATUS DSKI$ 関数 N C

| 機能 | ドライブに接続されているボリューム名を得ます． |
|---|---|
| 書式 | STATUS　DSKI$(<ドライブ名>) |
| 文例 | D$＝STATUS　DSKI$("B：") |

<ドライブ名>によって指定されたドライブに現在接続されている仮想ボリュームのボリューム名を得ます．ドライブに接続されているボリュームがないときはヌルストリング（空の文字列）が得られます．

参照：STATUS　DSKF

## STATUS ERROR 関数 P

| 機能 | モデムからのデータ受信時における通信エラーの有無を調べます． |
|---|---|
| 書式 | STATUS　ERROR〔(〔#〕<電話機番号>)〕 |
| 文例 | STATUS　ERROR(#1) |

<電話機番号>で指定した電話機がモデムからデータを受信したとき，エラーが発生したか否かを調べます．

<電話機番号>には，CMD　LINE　OPENで接続された電話機の番号を指定します．<電話機番号>が省略された場合は，1が指定されたものとみなされます．

エラーの状態は0〜7の数字で得らます．得られる数値とエラーの状態の対応は次のとおりです．

0　：　エラーなし
1　：　パリティエラー
2　：　オーバーランエラー
3　：　オーバーランエラー＋パリティエラー
4　：　フレーミングエラー
5　：　フレーミングエラー＋パリティエラー
6　：　フレーミングエラー＋オーバーランエラー
7　：　フレーミングエラー＋オーバーランエラー＋パリティエラー

注意：この関数はデータ通信モードでのみ使用可能です．

参照：STATUS　LINE，STATUS　MODE

## STATUS LINE 関数 P

| | |
|---|---|
| 機能 | 着信があったかどうかを調べます． |
| 書式 | STATUS　LINE〔(〔#〕〈電話機番号〉)〕 |
| 文例 | STATUS　LINE(#1) |

〈電話機番号〉で指定した電話に着信があったかどうかを調べます．

〈電話機番号〉には，CMD　LINE　OPEN で接続された電話機の番号を指定します．〈電話機番号〉が省略された場合は，1が指定されたものとみなされます．

着信があった場合には−1（真）が，着信がない場合には0（偽）が得られます．

注意：この関数を実行すると，着信があったという事象をクリアしてしまいますので，着信割り込みを使用するプログラムでこの関数を使用すると正しく動作しません．CMD　ON　LINE　GOSUB などと混在して使用しないようにしてください．

参照：CMD　ON　LINE　GOSUB，STATUS　ERROR，STATUS　MODE

## STATUS MODE 関数 P

| | |
|---|---|
| 機能 | モデム-NCU 内蔵電話機の現在のモードを調べます． |
| 書式 | STATUS　MODE〔(〔#〕〈電話機番号〉)〕 |
| 文例 | STATUS　MODE(#1) |

〈電話機番号〉で指定した電話機の現在のモードを調べます．

〈電話機番号〉には，CMD　LINE　OPEN で接続された電話機の番号を指定します．〈電話機番号〉が省略された場合は，1 が指定されたものとみなされます．

得られる数値と電話機のモードの対応は次のとおりです．

0 ： 初期モード（電話機が論理的に接続されていない状態）
1 ： オフラインコマンドモード
2 ： データ通信モード
3 ： 通話モード

電話機のモードの詳細に関しては「$N_{88}$-日本語 BASIC(86)(MS-MOS 版)ユーザーズマニュアル」を参照してください．

参照：CMD　LINE　OPEN，サンプルプログラム 40

## STATUS PLAY 関数　S C

機能　サウンドバッファの未演奏データのバイト数を得ます．
書式　STATUS　PLAY(〈チャンネル番号〉)
文例　PRINT　STATUS　PLAY(0)

〈チャンネル番号〉で指定されたチャンネルのサウンドバッファの未演奏データのバイト数を得ます．〈チャンネル番号〉と得られる値の関係は次のとおりです．

1〜6 ： 対応するチャンネルの未演奏データバイト数
0 ： すべてのチャンネルの未演奏データバイト数の中で最大のバイト数
負の数 ： すべてのチャンネルの未演奏データバイト数の中で最小のバイト数

参照：PLAY

## STOP　C

機能　プログラムの実行を一時中断し，コマンドモードにもどります．
書式　STOP
文例　STOP

STOP はプログラムの実行を一時中断するためのもので，プログラムのどこにおいてもかまいません．STOP を実行すると，次のメッセージが表示されます．

　　Break　in　〈行番号〉

ここで，〈行番号〉は実行の中断されたプログラム行の行番号を表します．

STOPが実行されると，BASICは常にコマンドレベルにもどります．また，CONTコマンドによりプログラムの実行を再開することができます．

STOPはENDとは異なり，そのときオープンされていたデータファイルをクローズしませんので注意してください．

コンパイラでコンパイルされたプログラムの場合，STOPが実行されると，次のメッセージが表示されます．

　　Press any key to command mode.

これに対し，任意のキーを押すと，プログラムの実行は終了し，MS-DOSに制御がもどります．

参照：CLOSE，CONT，END

# STOP　ON／OFF／STOP　C

**機　能**　STOPキーおよびCTRL＋Cによる割り込みの許可，禁止，停止を制御します．

**書　式**　1) STOP　ON
　　　　　2) STOP　OFF
　　　　　3) STOP　STOP

**文　例**　STOP　ON

**書式1)** 割り込みを許可します．以後STOPキーあるいはCTRL＋Cを押すごとに割り込みが発生し，ON　STOP　GOSUBによって設定された処理ルーチンに分岐します．

**書式2)** 割り込みを禁止します．以後STOPキーあるいはCTRL＋Cを押しても処理ルーチンへの分岐は起こりません（STOPキーあるいはCTRL＋Cは通常の動作になります）．

**書式3)** 割り込みを停止します．以後STOPキーあるいはCTRL＋Cを押しても押されたことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかし，その後，STOP　ONによって割り込みが許可されると，前にSTOPキーあるいはCTRL＋Cを押したことによって，処理ルーチンに分岐します．

**注意**：プログラムの終了時にはSTOP　OFFを実行しておいてください．

**参照**：ON　STOP　GOSUB

## STR$ 関数 C

**機能**　数値を文字列に変換します.

**書式**　STR$(〈数式〉)

**文例**　LISTING$＝STR$(NUM)＋NAME$

〈数式〉に指定した値を，文字列に変換します.

変換後の文字列の最初の文字は,〈数式〉の値が正ならば空白(スペース)となり,負ならば"－"(マイナス記号)となります.

〈数式〉には，整数型および実数型(倍精度，単精度)の数値を指定することができます.

注意：STRING$と区別すること.

参照：HEX$，OCT$，STRING$，VAL，サンプルプログラム24

## STRING$ 関数 C

**機能**　任意の文字を任意の数だけ連結した文字列を得ます.

**書式**　STRING$(〈式〉, 〈文字式〉 | 〈数式〉)

**文例**　A$＝STRING$(50,"＋")

〈文字式〉に指定した文字，または〈数式〉に指定したキャラクタコードに該当する文字を，〈式〉に指定した数だけ連結した文字列を得ます.

〈文字式〉の場合には最初の1文字のみが用いられます.〈数式〉の場合は，値が0から255の範囲内でなくてはなりません.

参照：STR$

## SWAP　Ⓒ

| 機　能 | 2つの変数の値を入れ換えます. |
|---|---|
| 書　式 | SWAP　〈変数〉,〈変数〉 |
| 文　例 | SWAP　A$, B$ |

どの型(整数型, 単精度型, 倍精度型, または文字型)の変数でも, SWAPによってその値を交換することができます.

ただし, 2つの変数の型が一致していないと, "Type　mismatch" エラーが起こります.

参照：サンプルプログラム10

## SYSTEM　Ⓒ

| 機　能 | MS-DOSに制御をもどします. |
|---|---|
| 書　式 | SYSTEM |
| 文　例 | SYSTEM |

プログラムの実行を終了し, MS-DOSに制御をもどします.

SYSTEMが実行されると, 開かれているファイルはすべて閉じられ, またメモリ中にあるプログラムや変数などはすべてクリアされてからMS-DOSに制御がもどります. 必要なプログラムはあらかじめ必ずディスクにセーブしておいてください.

コンパイラによってコンパイルされたプログラムの場合は, ENDの実行によってもMS-DOSに自動的に制御がもどります.

参照：END

## TAB 関数　Ⓒ

| 機　能 | 出力対象行の任意の位置まで空白を出力します. |
|---|---|
| 書　式 | TAB(〈数式〉) |
| 文　例 | PRINT　"SUPER"；TAB(10)；"BASIC" |

出力の対象となる行の行頭から, 〈数式〉に指定した桁数分だけ, 空白(スペース)を空けます.

ただし, TABは, PRINTやLPRINTなどの出力文中でのみ使用することができるもので,

文字式としては使えません.

〈数式〉には,−32768から32767までの範囲の数値を指定できますが,負の数はすべて0とみなされます.また,正の数の場合でも,WIDTHで決められている現在の水平表示桁数以上の場合は,〈数式〉をその表示桁数で割った余りを値とします.

TABでは,SPCと違い,〈数式〉で指定する数は常に行頭からの桁数,つまり位置を表しますから,作表などに使用すると便利です.

参照:SPC,サンプルプログラム22,30

## TAN 関数 C

| 機 能 | 正接(タンジェント)を得ます. |
|---|---|
| 書 式 | TAN(〈数式〉) |
| 文 例 | Y=X＊TAN(ANGLE) |

〈数式〉の値に対する正接値を得ます.〈数式〉の単位はラジアンで指定します.

〈数式〉に倍精度実数が含まれる場合,得られる値は倍精度となりますが,他の場合には単精度となります.

参照:ATN,COS,SIN

## TIME$ 関数 C

| 機 能 | 時刻を得ます. |
|---|---|
| 書 式 | 1) TIME$ |
| | 2) TIME$="hh : mm : ss" |
| 文 例 | PRINT　TIME$ |
| | TIME$="23 : 15 : 00" |

TIME$には常に現在の時刻が"hh:mm:ss"(時:分:秒)の形で入れられており,いつでもその内容を見ることができます.時刻表示は,24時制です.

なお,書式2)を用いることにより,時刻を変更することもできます.hhには00〜23,mmには00〜59,ssには00〜59の範囲の整数文字列を指定します.hh,mm,ssの間はコロン(:)で区切ります.

注意:時刻は,バッテリーバックアップによって自動的に更新され,正しく維持されるように

なっています．不用意に時刻を変えないようにしてください．

参照：DATE$，サンプルプログラム31

## TIME$ ON／OFF／STOP　C

機能　リアルタイムタイマによる割り込みの許可，禁止，停止を制御します．

書式　1）TIME$　ON

2）TIME$　OFF

3）TIME$　STOP

文例　TIME$　ON

書式1）割り込みを許可します．以後，設定された時刻になると割り込みが発生し，ON TIME$　GOSUBによって設定された処理ルーチンに分岐します．

書式2）割り込みを禁止します．以後，設定された時刻になっても処理ルーチンへの分岐は起こりません．

書式3）割り込みを停止します．以後，設定された時刻になってもそのことを覚えているだけで，処理ルーチンへの分岐は起こりません．しかしTIME$　ONによって割り込みが許可されると，先ほどの割り込みで処理ルーチンに分岐します．

注意：プログラムの終了時にはTIME$　OFFを実行しておいてください．

参照：ON　TIME$　GOSUB，TIME$，サンプルプログラム31

## TRON／TROFF　C

機能　プログラムの実行状態を追跡します．

書式　1）TRON

2）TROFF

文例　TRON

TROFF

TRONを実行してからプログラムをRUNさせると，以後実行しているプログラムの行番号がカギカッコ（[]）つきでテキスト画面に表示され続けます．

TRON状態を中止するには，TROFFを実行します．なお，LOAD，NEWを実行した場合もTRON状態は解除されます．

## USR 関数 C

| | |
|---|---|
| 機 能 | メモリ上に用意された機械語関数を呼び出します. |
| 書 式 | USR〔<番号>〕(<引数>) |
| 文 例 | I=USR3(J) |

呼び出したい機械語関数はあらかじめメモリ上に用意しておき,またDEF　USRにより,その実行開始アドレスを設定しておかなくてはなりません.機械語関数を用意するには,POKEやBLOADを用いることができます.

〈番号〉には,DEF　USRにより定義された番号を0から9までの値で指定します.省略された場合には0と解釈されます.

〈引数〉には,BASICから機械語関数に渡す変数,定数,式を指定します.

USRが実行されると,直前に実行されたDEF　SEGで指定されたセグメントベースに,DEF　USRにより定義された機械語関数の実行開始アドレス(相対アドレス値)が加えられた番地に実行が移されます.USRによって呼び出された機械語関数は,機械語のIRET命令によりBASICに制御をもどすことができます.

なお,機械語プログラムを呼び出す方法としては,USRの他にCALLが用意されています.CALLでは複数の引数を機械語プログラムに受け渡すことができます.

参照:BLOAD,CALL,CLEAR,DEF　USR,POKE,SEGPTR

## VAL 関数 C

| | |
|---|---|
| 機 能 | 文字列表記の数値を実際の数値に変換します. |
| 書 式 | VAL(<文字列>) |
| 文 例 | A=VAL("&H20") |

〈文字列〉に指定した文字列表記の数値を,実際の数値に変換します.

〈文字列〉には,整数表記(8進形式,10進形式,16進形式)および実数表記(通常の小数点形式,指数形式)のいずれも指定できます.

〈文字列〉の最初の文字が+,-,&,または数字でなければ,VALの値は0になります.また,〈文字列〉中に数字を表す文字以外の文字が含まれていると,その文字以降の文字は無視されます.ただし,16進表記ならばA～Fも数字とみなされ,8進表記ならば8および9は数字とみなされないことに注意してください.なお,文字列中のスペースは無視されます.

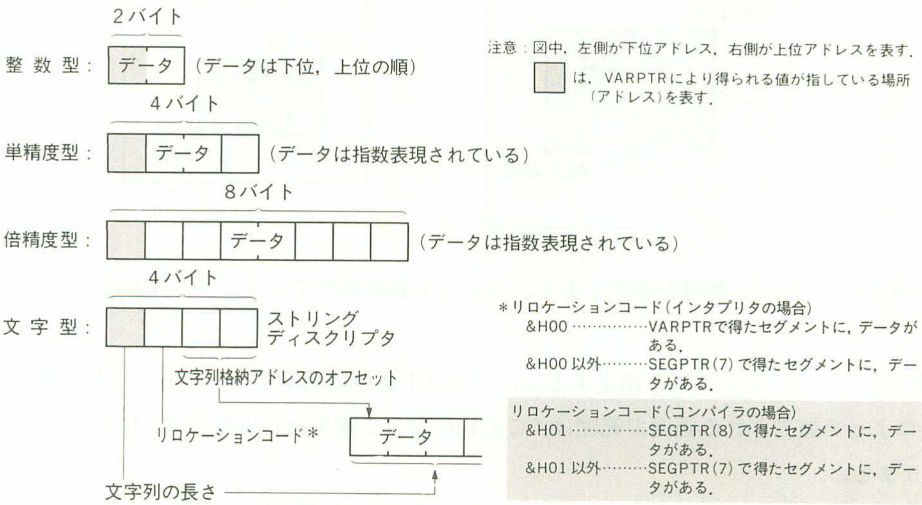
参照:STR$

# VARPTR 関数 C

| | |
|---|---|
| 機能 | 変数の値が格納されているメモリ番地，ファイルに割り当てられているファイルコントロールブロックの開始番地を得ます． |
| 書式 | 1）VARPTR(〈変数名〉〔，〈機能〉〕)<br>2）VARPTR(#〈ファイル番号〉〔，〈機能〉〕) |
| 文例 | PRINT　HEX$(VARPTR(A，1))<br>PRINT　HEX$(VARPTR(A))<br>SEGM%＝VARPTR(#1，1)<br>ADDR%＝VARPTR(#1) |

書式1）〈変数名〉で指定した変数あるいは配列変数のデータが格納されている領域のメモリ番地を得ます．

〈機能〉に0を指定すると相対アドレスが得られ，1を指定するとセグメントベースが得られます．省略した場合は，0が指定されたものとみなされます．




書式2）指定した〈ファイル番号〉に割り当てられているファイルコントロールブロックの開始番地を得ます．

〈機能〉に0を指定すると相対アドレスが得られ，1を指定するとセグメントベースが得られます．省略した場合は，0が指定されたものとみなされます．

なお，ファイルに割り当てられている入出力バッファの開始番地は，このファイルコントロールブロックの&H61番地(相対アドレス)の2バイトに，SEGPTR(7)で得られるセグメントベースからの相対アドレスの形式で格納されています．

参照：DEF SEG，SEGPTR

# VIEW

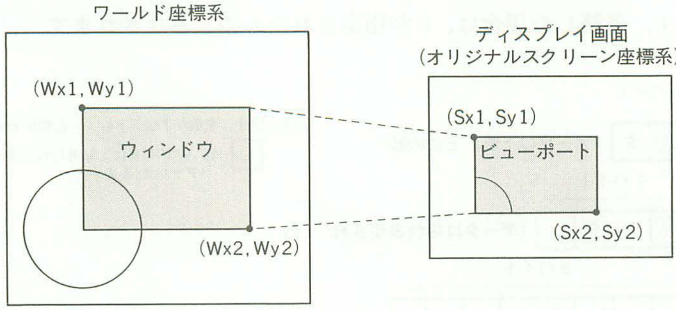
C

| 機能 | ディスプレイ画面上での表示領域(ビューポート)を指定します. |
|---|---|
| 書式 | VIEW (Sx1, Sy1)－(Sx2, Sy2)〔, 〈領域色〉〕〔, 〈境界色〉〕 |
| 文例 | VIEW (100, 30)－(200, 75),, 7 |

オリジナルスクリーン座標系上の(Sx1, Sy1)を左上の頂点, (Sx2, Sy2)を右下の頂点とする長方形を図形表示領域として指定します. ここでいうオリジナルスクリーン座標とは, ディスプレイ画面のドットと1対1に対応して, WINDOW, VIEWにより変化することのない物理的な座標のことです.

VIEWが実行されると, WINDOWにより指定されているワールド座標系上の領域内の図形は, VIEWで指定した領域中に表示されるようになります.

この領域をビューポート(View Port)といいます.




〈領域色〉にパレット番号を指定すると, パレット番号の色でビューポート内をぬりつぶします.

〈境界色〉にパレット番号を指定すると, パレット番号の色でビューポートの枠を描きます. CLSによって消去される画面の範囲は, この枠によって囲まれた中だけで, 枠は消されることはありません.

VIEWは, 図形の表示領域の指定を行うだけで, 実際の画面に対する操作は行いませんので, VIEWによりビューポートを変更することによって, 以前のビューポート中の図形が移動するようなことはありません. また, VIEWはすべてのグラフィック画面表示の対象範囲を指定された領域に限定してしまいますから, ビューポートの外に点や線などを表示することはできなくなります.

VIEWの指定を変えることにより,1つの図形を描くプログラムでも,画面上の異なる位置に異なる大きさで図形を描くことができます.

Sx1 <Sx2, Sy1 <Sy2が成り立たない場合, あるいはこれらの座標がディスプレイ画面から外れている場合には, "Illegal function call" エラーとなります.

一度設定されたビューポートは, 次にVIEWあるいはSCREENが実行されるまで変化しません. また, VIEWはLP(最終参照点)をビューポートの左上の頂点に移動します.

**注意**: ワールド座標系がビューポート内に展開されるのはWINDOWの実行後であり, VIEWのみ実行してウィンドウは初期状態のままであれば, ビューポート内の座標は, (ワールド座標)=(スクリーン座標)となります.

**参照**: CLS, SCREEN, VIEW(関数), WINDOW, WINDOW(関数), サンプルプログラム20, 21

## VIEW 関数 Ⓒ

| | |
|---|---|
| 機能 | **現在のビューポートの設定位置を得ます.** |
| 書式 | VIEW(〈機能〉) |
| 文例 | SX1=VIEW(0) |

VIEWにより設定されている現在のビューポートの設定位置(Sx1, Sy1およびSx2, Sy2)を, 〈機能〉に指定する値により, それぞれ個別に得ます.

〈機能〉には0〜3の数値を指定します.

- 0 : ビューポートの左上の頂点のX座標(Sx1)
- 1 : ビューポートの左上の頂点のY座標(Sy1)
- 2 : ビューポートの右下の頂点のX座標(Sx2)
- 3 : ビューポートの右下の頂点のY座標(Sy2)

**注意**: VIEW関数は, ディスプレイ画面上のビューポートの位置を得る関数ですから, 得られる値は, スクリーン座標系の値となります.

**参照**: MAP, VIEW, WINDOW, WINDOW(関数)

# VOICE

S C

| | |
|---|---|
| 機 能 | FM音源の音色バンクを再定義します． |
| 書 式 | VOICE 〈音色番号〉，〈整数型配列名〉 |
| 文 例 | VOICE 5，A% |

FM音源の音色を自由に設定するための命令です．

サウンド拡張命令では，FM音源で最大82種類の音色を使用することができます．これらの音色は0～81の番号で管理されており，初期状態では内蔵音色のデータが各音色番号に割り当てられています．82種類のうち，0番のデフォルト音色以外はすべて，再定義することによって自分の好みの音色を割り当てることができます．

音色の再定義は，音色番号の音色パラメータ（音色のデータ）を別の音色パラメータに変えることによって行います．音色番号と音色パラメータを合わせて音色バンクといいます．

〈音色番号〉には，音色を再定義したい音色番号（1～81）を指定します．

〈整数型配列名〉には，新しく与える音色パラメータを格納した整数型配列変数の配列変数名を指定します．配列変数にはあらかじめ音色パラメータを指定しておかなければなりません．

配列変数の各要素と音色パラメータとは，次のように対応しています（配列名は，仮にVOICE%とします）．

ここで，OPは各オペレータの番号を表し，1～4を指定します．

| | | |
|---|---|---|
| VOICE%（0，0） | ： | フィードバック／アルゴリズム |
| VOICE%（0，1） | ： | オペレータマスク |
| VOICE%（0，2） | ： | LFO波形 |
| VOICE%（0，3） | ： | LFO　SYNCディレイ |
| VOICE%（0，4） | ： | LFO速さ |
| VOICE%（0，5） | ： | LFOピッチ変調深さ（微調整） |
| VOICE%（0，6） | ： | LFO振幅変調深さ（微調整） |
| VOICE%（0，7） | ： | LFOピッチ変調深さ（粗調整） |
| VOICE%（0，8） | ： | 未使用 |
| VOICE%（0，9） | ： | 未使用 |
| VOICE%（OP，0） | ： | アタック系数 |
| VOICE%（OP，1） | ： | ディケイ系数 |
| VOICE%（OP，2） | ： | サスティン系数 |
| VOICE%（OP，3） | ： | リリース系数 |
| VOICE%（OP，4） | ： | サスティンレベル |
| VOICE%（OP，5） | ： | 出力レベル |

VOICE% (OP, 6)　：　キーボードレイトスケーリング深さ
VOICE% (OP, 7)　：　マルチプル
VOICE% (OP, 8)　：　デチューンレイト
VOICE% (OP, 9)　：　LFO振幅変調深さ

再定義された音色はPLAY　ALLOC, CLEAR, VOICE　INITが実行されるまで有効です.

**注意**：この命令やVOICE　COPYで使用する配列は, 5×10の大きさをもち, OPTION BASEが0のものでなくてはなりません. したがって, プログラムの初めに,

```
OPTION BASE 0
DIM VOICE%(4,9)
```

のように宣言しておきます.

VOICEで配列を使用した後はERASEを実行しないでください. ERASEを行うとサウンド拡張命令の動作は保証されなくなります. ERASEを行う場合は, PLAY　ALLOC, VOICE　INITなどを実行してからにしてください. 詳細については, サウンドボードに添付されている「サウンドボードユーザーズマニュアル」を参照してください.

**参照**：PLAY, PLAY　ALLOC, VOICE　COPY, VOICE　INIT

# VOICE COPY　S C

**機　能**　FM音源の音色パラメータを配列にコピーします.
**書　式**　VOICE　COPY　〈音色番号〉, 〈整数型配列名〉
**文　例**　VOICE　COPY　4, PARA%

〈音色番号〉で指定されたFM音源の音色番号の音色パラメータを, 〈整数型配列名〉にコピーします.

〈音色番号〉は0～81の値で指定します. 〈整数型配列〉の内容はVOICEと同じ構成になっています.

**参照**：VOICE, VOICE　INIT

# VOICE　INIT　S C

機　能　FM 音源の音色番号と音色パラメータを初期化します．

書　式　VOICE　INIT

文　例　VOICE　INIT

FM 音源の音色バンク(音色番号と音色パラメータ)をすべて初期化します．

VOICE で使用した音色パラメータ格納用の整数配列変数は，この命令の実行により，通常の配列変数となります（ERASE を行っても，別の用途に使用してもかまいません)．

注意：この命令を実行しても，SSG 音源は初期化されません．

参照：VOICE，VOICE　COPY

# VOICE　LFO　S C

機　能　各チャンネルの出力に LFO 効果を与えます．

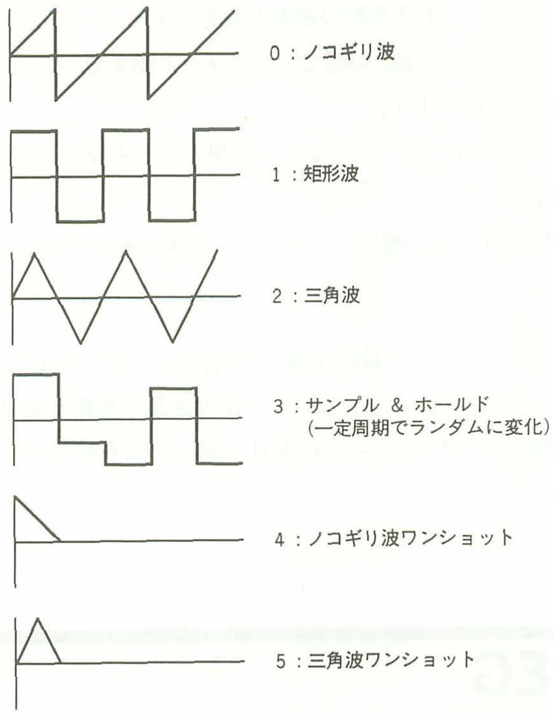
書　式　VOICE　LFO　<チャンネル番号>〔，<波形>〕〔，<SYNC ディレイタイム>〕〔，<速さ>〕〔，<ピッチ変調深さ(微)>〕〔，<振幅変調深さ>〕〔，ピッチ変調深さ(粗)>〕

文　例　VOICE　LFO　2，3，10，1500，－15，0，7

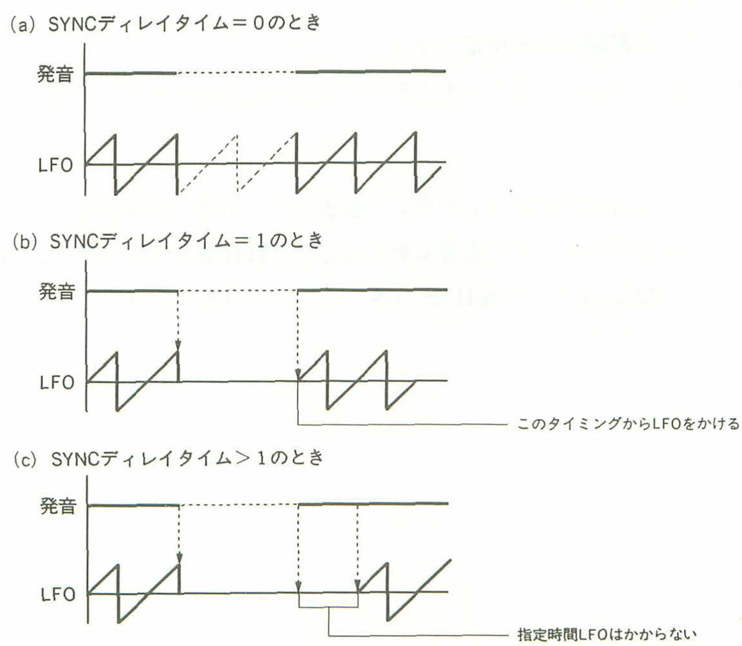
<チャンネル番号> で指定されたチャンネルの音にビブラート，トレモロなどの LFO 効果をつけます．

<波形> は 0～5 の範囲で，LFO の変調波形を次の中から選びます．

0 ： ノコギリ波
1 ： 矩形波
2 ： 三角波
3 ： サンプル&ホールド
　　(一定周期でランダムに変化)
4 ： ノコギリ波ワンショット
5 ： 三角波ワンショット




〈SYNCディレイタイム〉は0〜255の値で，音が出てからLFO効果が現れるまでの時間を設定します．0の場合は，音の出るタイミングと関係なく，非同期にLFO効果を付加します．1〜255の場合は，音が出てから〈SYNCディレイタイム〉で指定した時間だけ経過した後に同期してLFO効果を付加します．



〈速さ〉は0〜16383の値で，LFO波形の速度を設定します．

〈ピッチ変調深さ(微)〉は−127〜127の値で，ビブラート効果のかかる深さを設定します．値が負のときはLFO波形は反転します．

〈振幅変調深さ〉は−127〜127の値で，トレモロ効果のかかる深さを設定します．値が負のときはLFOの波形は反転します．

〈ピッチ変調深さ(粗)〉は0〜15の値で，ビブラート効果のかかる深さをおおまかに設定します．

**注意**：〈波形〉に4，5のワンショット波形を選んだ場合は，必ず〈SYNCディレイタイム〉を1以上の値にしてください．そうしないと，出力される音程や音量が不定となります．なお，〈振幅変調深さ〉のパラメータはFM音源にのみ有効です．

**参照**：PLAY

# VOICE　REG

S C

| | |
|---|---|
| 機　能 | シンセサイザLSIのレジスタに値を設定します． |
| 書　式 | VOICE　REG　〈レジスタ番号〉，〈式〉 |
| 文　例 | VOICE　REG　&H28，240 |

〈レジスタ番号〉には&H00〜&HB2（0〜178）の値で，書き込むべきシンセサイザLSIの内部レジスタ番号を指定します．

〈式〉には0〜255の範囲の値を指定します．

この命令はMML（ミュージックマクロランゲージ）のYr，dコマンドと同じ機能をもっています．

**注意**：シンセサイザLSIの内部には，サウンド拡張命令の制御に直接関係するものもありますので，この命令でレジスタを直接変更するときには注意してください．特にレジスタ番号&H21〜&H27，&H2D〜&H2Fの各レジスタへの書き込みは行わないでください．

**参照**：PLAY

# WAIT Ⓒ

| 機能 | 入力ポートをモニタする間，プログラムの実行を停止します． |
|---|---|
| 書式 | WAIT 〈ポート番号〉，〈式 1〉〔，〈式 2〉〕 |
| 文例 | WAIT 1，&H22，&HFF |

WAIT は，指定した入力ポートのビットパターンが指定した状態になるまでプログラムの実行を停止します．

〈ポート番号〉は入力ポートの番号で，0〜32767(&H0〜&H7FFF)の範囲内の数値で指定します．

WAIT は，まずポートから読み込んだデータと〈式 2〉との XOR をとり，次にその結果と〈式 1〉の AND をとります．もしその結果が 0(偽)なら，BASIC はもう一度ポートの状態を読み込み，同じ操作を繰り返します．もし結果が 0 でない(真)ならば，プログラムの実行は次の文に移ります．〈式 2〉を省略した場合は 0 とみなされます．

注意：WAIT の実行により，プログラムの実行が無限ループに入ってしまう場合があります．
　　その場合にはコンピュータをリセットしなければなりません．

参照：INP，OUT

# WBYTE Ⓖ Ⓒ

| 機能 | マルチラインメッセージおよびバイナリデータを送出します． |
|---|---|
| 書式 | 1）WBYTE 〔〈コマンド〉〔，〈コマンド〉〕〕；〔〈データ〉〔，〈データ〉〕〕〔@〕<br>2）WBYTE；〈データ〉〔，〈データ〉〕〔@〕 |
| 文例 | WBYTE &H5F； |

書式 1）マスタモードの場合に使用します．

ATN（アテンション）を true にして〈コマンド〉を送出した後，ATN を false にして〈データ〉を送出します．

命令の最後に〔@〕を指定した場合は，最後のデータバイトの出力時に EOI が true になります．

〈コマンド〉および〈データ〉は 0〜255（&H00〜&HFF）までのバイナリデータを指定します．

書式 2）スレーブモードの場合に使用します．

この命令が実行されると，コントローラからトーカとしてアドレスされるまで待ちます．

MTA（マイトークアドレス）を受信後，ATN が false になると，バイナリデータを送出します．

命令の最後に〔@〕が指定した場合は，最後のデータバイトの出力時に EOI が true になります．

スレーブモードでは，マルチラインメッセージの送出はできないため，〈コマンド〉はすべて省略しなければなりません．

参照：RBYTE，PRINT@

## WHILE～WEND C

機　能　WHILE から WEND までの区間中にある一連の文を，指定条件が満足されている間，繰り返して実行します．

書　式　WHILE 〈論理式〉
　　～
WEND

文　例　WHILE J=<5
　　～
WEND

WHILE～WEND ループ中(WHILE と WEND の間)におかれた文を，〈論理式〉が真である(0 でない)間，繰り返して実行します．〈論理式〉が偽(0)になると繰り返しは終わり，WEND に続く文に制御が移ります．

最初から〈論理式〉が偽の場合は，WHILE と WEND との間におかれた文は一度も実行されません．

WHILE～WEND は FOR～NEXT と同じ入れ子構造にすることができます．この場合には，それぞれの WEND はそれ以前の最も近くにある WHILE と対応します．WEND は WHILE と対になり，ループの終わりを示す働きをしますので省略することはできません．

注意：WHILE～WEND は必ず 1 対 1 に対応していなければなりません．
また，WHILE～WEND ループ内へ外部から GOTO などでジャンプして入ってきたり，逆にループ内から外部へジャンプしたりするようなプログラムは，その動作が保証されなくなります．

参照：FOR…TO…STEP～NEXT，サンプルプログラム11

# WIDTH Ⓒ

| | |
|---|---|
| 機　能 | 各種入出力機器やファイルに対して1行の長さなどを指定します. |
| 書　式 | 1) WIDTH 〈桁数〉〔,〈行数〉〕 |
| | 2) WIDTH 〈ファイルディスクリプタ〉,〈サイズ〉 |
| | 3) WIDTH #〈ファイル番号〉,〈サイズ〉 |
| 文　例 | WIDTH 80, 25 |
| | WIDTH "LPT1:", 80 |
| | WIDTH #1, 100 |

**書式1)** テキスト画面に表示する文字の量を指定します.〈桁数〉は1行あたりの文字数を表し,40文字か80文字かのどちらかを指定します. また〈行数〉は1画面あたりの行数を表し,20行か25行かのどちらかを指定します.〈行数〉を省略した場合には,そのときの行数のままで変化しません.

**書式2)** 〈ファイルディスクリプタ〉で指定したデバイスファイルに対して,1行の長さを指定します. ここで指定可能なデバイスファイルは,RS-232C回線ファイルとプリンタです.〈サイズ〉の値は0から255まで許されています. 0が256と解釈される以外は,指定した値が1行の文字数となります. たとえば,この命令がプリンタに対して実行されたとすれば,プリンタの1行に印字される文字数は〈サイズ〉で指定したものとなります. このようにプリンタに対して実行した場合,WIDTH　LPRINTと同等の機能を得ることができます. 初期状態では255に設定されています.

**書式3)** 〈ファイル番号〉に割り当てられているバッファ(OPEN参照)に対して,その大きさを〈サイズ〉で指定します. ここで指定できるデバイスファイルは,RS-232C回線ファイルとプリンタです.〈サイズ〉は0から255まで許されています. 0が256と解釈される以外は,指定した値が1行の文字数となります. 以後このファイル番号とデバイスとの入出力はこの〈サイズ〉単位で行われます. 初期状態では255に設定されています.

なお,RS-232C回線ファイルに対してWIDTHを実行すると,ファイルに対するデータの送出時,〈サイズ〉で指定された桁数の位置ごとに改行(CR)コードを送出することになります.

**注意**:2バイト系日本語1文字は文字数2文字(バイト)として数えます.

**参照**:WIDTH　LPRINT,サンプルプログラム33

# WIDTH LPRINT　Ⓒ

機　能　プリンタに出力する1行あたりの文字数を設定します.

書　式　WIDTH　LPRINT　〈文字数〉

文　例　WIDTH　LPRINT　80

プリンタに印字する場合の，1行あたりの文字数を設定します.〈文字数〉には0から255までの値を指定できます.0が256と解釈される以外は，指定した値が1行の文字数となります.

注意：2バイト系日本語1文字は文字数2文字(バイト)として数えます.

参照：WIDTH

# WINDOW　Ⓒ

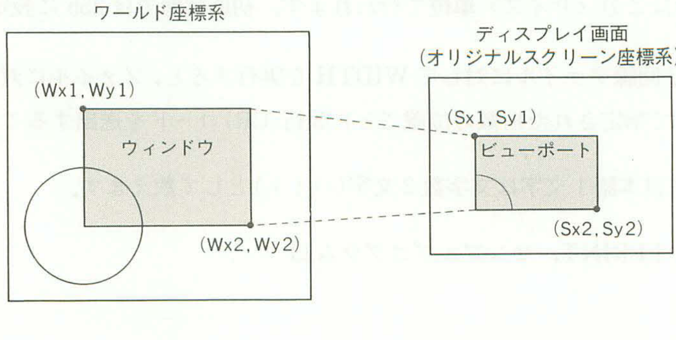
機　能　ビューポートに表示するワールド座標系内の領域を指定します.

書　式　WINDOW　(Wx1，Wy1)－(Wx2，Wy2)

文　例　WINDOW　(－300，－50)－(100，70)

ワールド座標系上の(Wx1，Wy1)を左上の頂点，(Wx2，Wy2)を右下の頂点とする長方形で囲まれた領域を，ディスプレイ画面上のビューポート(VIEWで指定した領域)内に表示するよう設定します.

このようにして指定された領域はウィンドウと呼ばれ，ウィンドウから外れたところに描かれる図形は表示されなくなります.ウィンドウの大きさを変えると，対象領域の大きさが変わりますから，同じグラフィックス命令で描かれる図形でも，ビューポート内では異なった大きさで表示されることになります.




WINDOWも，VIEWと同じように領域の指定を行うだけで実際の画面に対する操作は行い

ませんので，WINDOWによってウィンドウの設定を変更するだけで，ビューポート中に表示される図形が変化するようなことはありません．また，図形が何も描かれない領域をウィンドウとして設定すると，ディスプレイ画面上のビューポートには何も描かれないことになりますので，注意してください．

Wx1<Wx2, Wy1<Wy2が成り立たない場合，あるいはこれらの座標がワールド座標系から外れている場合には，〝Illegal function call〞エラーとなります．WINDOWで指定する座標は，ワールド座標ですから，負の値や実数値もとることができます．

一度設定されたウィンドウは，次にWINDOWあるいはSCREENが実行されるまで変化しません．またWINDOWはLP(最終参照点)をウィンドウの左上の頂点に移動します．

注意：ワールド座標系がビューポート内に展開されるのはWINDOWの実行後であり，VIEWのみ実行してウィンドウは初期状態のままであれば，ビューポート内の座標は，(ワールド座標)＝(スクリーン座標)となります．

参照：VIEW，VIEW(関数)，WINDOW(関数)

# WINDOW 関数 C

| | |
|---|---|
| 機能 | 現在のウィンドウの設定位置を得ます． |
| 書式 | WINDOW(〈機能〉) |
| 文例 | WY1＝WINDOW(1) |

WINDOWにより設定されている現在のウィンドウの位置(Wx1，Wy1およびWx2，Wy2)を，〈機能〉に指定する値により，それぞれ個別に得ます．

〈機能〉には0〜3の数値を指定します．

0 ： ウィンドウの左上の頂点のX座標(Wx1)
1 ： ウィンドウの左上の頂点のY座標(Wy1)
2 ： ウィンドウの右下の頂点のX座標(Wx2)
3 ： ウィンドウの右下の頂点のX座標(Wy2)

注意：WINDOW関数で得られる値は，ワールド座標系の値となります．

参照：MAP，VIEW，VIEW(関数)，WINDOW

# WRITE

**C**

**機　能**　画面にデータを出力します．

**書　式**　WRITE 〈式〉〔 , | ; 〈式〉…〕

**文　例**　WRITE　LISTNUM, GOODS$, COST

〈式〉に指定された数値式あるいは文字式の値を，画面に出力します．

〈式〉はコンマ(,)あるいはセミコロン(；)により区切ります．PRINTと違い，コンマとセミコロンとの機能上の区別はありません．

WRITEは，ほぼ，PRINTと同じように式の値を出力しますが，不要な空白桁は詰め，それぞれの式の値の間は必ずコンマで区切ります．また，文字列はダブルクォーテーション(")で囲んで出力します．

WRITEは，各式の値を出力したのち，改行を行います．

参照：PRINT，WRITE #，サンプルプログラム12

# WRITE #

**C**

**機　能**　ファイルにデータを書き出します．

**書　式**　WRITE　#〈ファイル番号〉，〈式〉〔 , | ; 〈式〉…〕

**文　例**　WRITE　#1, LISTNUM, GOODS$, COST

出力モードでオープンしたファイル(シーケンシャルファイル，スクリーンファイル，プリンタなど)や，RS-232C回線ファイルに，〈式〉により指定した文字列，数値などのデータを書き出します．

〈ファイル番号〉には，OPENによって，そのファイルをオープンしたときに使った番号を指定します．

〈式〉を複数指定する場合は，コンマ(,)あるいはセミコロン(；)により区切ります．PRINT #と違い，コンマとセミコロンとの機能上の区別はありません．

WRITE #は，ほぼ，PRINT #と同じように式の値を出力しますが，不要な空白桁は詰め，それぞれの式の値の間は必ずコンマで区切ります(コンマそのものをデータとして出力します)．また，文字列はダブルクォーテーション(")で囲んで出力します．

WRITE #は，各式の値を出力したのち，改行コード(CHR$(13))を書き出します．

WRITE #は，区切り記号としてコンマを必ず出力し，不要な空白の出力は行わないため，PRINT #に比べファイルの使用領域を節約することができます.

**注意**：出力対象ファイルが "SCRN：" の場合，日本語データを出力することはできません.

**参照**：PRINT #，WRITE

# 第3章

# サンプルプログラム

# 第3章　サンプルプログラム

## この章の見方

この章には，$N_{88}$-BASIC (86)の各種命令・関数を使用したサンプルプログラムが掲載されています．

それぞれのプログラムは，相互に関連のある命令・関数を使って構成してあります．また，ポイントとなる各命令・関数には，右側に解説がつけてあります．

なお，各プログラムの番号は，「第２章　命令リファレンス」中の各命令の〝参照〟部分に示されたサンプルプログラム番号と対応しています．

以下に，本章中に掲載されているサンプルプログラムの番号・標題と，各プログラム中で使用されている主な命令の，一覧を示します．

16．パレット（4096色中・16色モード）
〔1〕COLOR, LINE, 〔2〕COLOR

17．DRAW
POINT, DRAW, PSET, PRESET,
LINE, 〔1〕PAINT

18．グラフィックパターンの読み込みと表示
GET@, PUT@, LINE, CIRCLE

19．グラフィック画面のスクロール
PUT@, ROLL

20．ビューポートの設定
SCREEN, WINDOW, VIEW, LINE

21．ウィンドウの設定
SCREEN, WINDOW, VIEW, CIRCLE

22．エラー処理ルーチン
ON ERROR GOTO, ERROR, ERR,
RESUME

23．文字とキャラクタコードの変換
ASC, MID$, CHR$

24．８進変換と16進変換
OCT$, HEX$, STR$

25．文字列の抜き出し
LEFT$, RIGHT$, MID$, LEN, SPACE$

26．数値データの文字型化
MKI$, MKS$, MKD$, OPEN, CLOSE,
FIELD, LSET, PUT

27．文字型化数値データの数値化
CVI, CVS, CVD, OPEN, CLOSE,
FIELD, GET

28．シーケンシャルファイル
OPEN, CLOSE, PRINT #, INPUT #,
EOF

29．ランダムファイル
OPEN, FIELD, CLOSE, LOF, GET,
LSET, PUT, INPUT$, INKEY$

30．キー割り込みルーチン
ON KEY GOSUB, KEY ON,
RETURN, SIN, COS

31．タイマー割り込みルーチン
TIME$, TIME$ ON,
ON TIME$ GOSUB, RETURN, BEEP

32．乱数の発生
RANDOMIZE, RND

33．テキスト画面のモード設定と文字に対する色・機能の設定
CONSOLE, WIDTH, LOCATE,
COLOR@, GOSUB, RETURN

34．利用者定義文字の登録
KPLOAD, KNJ$

35．２バイト系日本語文字列の変換と抜き出し
AKCNV$, KACNV$, KMID$

36．２バイト系日本語文字列と漢字コードの 相互変換
JIS$, KNJ$, KMID$

37．特定タイプの文字列の抜き出し
KEXT$

38．２バイト系日本語文字を含む文字列の長さとタイプ

KLEN, KTYPE

39．利用者定義関数

DEF FN

40．電話制御機能（拡張機能）

CMD ERROR ON，CMD LINE ON，
CMD LINE OPEN，CMD MODE CUT，
CMD ON ERROR GOSUB，
CMD ON LINE GOSUB，
CMD RECEIVE，STATUS MODE

41．サウンド機能（拡張機能）

ON PLAY GOSUB，PLAY，
PLAY ALLOC，PLAY ON

42．10進演算機能

PCAL$，PCHK，PCNV$

## 1．プログラムの連結(その1)

```
100 DIM IDENT(200)
110 COMMON IDENT(),COUNT ──────── 引き渡す変数を宣言.
120 OPEN "DATA" AS #1 ──────── "DATA"ファイルをランダムモードでオープン.
130 FIELD #1,4 AS I$,20 AS N$,50 AS A$ ── 会員番号，氏名，住所の各フィールド変数を定義.
140 INPUT "会員番号(1-999)";I
150 IF I=0 THEN GOTO *PRINTOUT
160 IF I=>1000 THEN STOP
170 INPUT "氏名";NAME1$
180 INPUT "住所";ADR$
190 LSET I$=MKS$(I) ─┐
200 LSET N$=NAME1$   ├──── 各データを各フィールド変数にセット.
210 LSET A$=ADR$ ────┘
220 PUT #1,I ──────── 会員番号と同番のレコードにデータを書き出す.
230 COUNT=COUNT+1
240 IDENT(COUNT)=I ──────── 書き出された会員番号(レコード番号)を憶えておく.
250 PRINT:GOTO 140
260 *PRINTOUT
270 CHAIN "CHAIN.S02" ──────── "CHAIN.S02"に実行を移す.
280 END                         "DATA"ファイルはオープンしたまま.
```

## 2．プログラムの連結(その2)

```
100 COMMON INDENT(),COUNT ──────── 引き渡される変数を宣言.
110 PRINT:PRINT
120 FIELD #1,4 AS I$,20 AS N$,50 AS A$ ── "DATA"ファイルはOPEN状態のままCHAIN
130 IF COUNT=0 THEN PRINT "END":END        されるので，新たにOPENしなくてよい.
140 GET #1,IDENT(COUNT) ──────── 実際に書き出されているレコードのみを読み込む.
150 PRINT "会員番号:";CVS(I$)
160 PRINT "氏名:";N$
170 PRINT "住所:";A$
180 COUNT=COUNT-1
190 PRINT:GOTO 130
```

## 3．DATA行の読み込み

```
100 READ A,B,C
110 RESTORE ──────── 最初のDATA行(190行)から読み出すように指定.
120 READ D,E,F
130 RESTORE *STRDATA ──── *STRDATA行(210行)以降のDATA行(220行)から読み出すように指定.
140 READ A$,B$,C$
150 PRINT A,B,C
160 PRINT D,E,F
170 PRINT A$,B$,C$
180 END
190 DATA 1,2,3
200 DATA 4,5,6 ──────── このプログラムでは，結局このDATA行は読み出されない.
210 *STRDATA
220 DATA AA,BB,CC
```

## 4．変数の型宣言

```
100 DEFINT J-M ──────── J～Mで始まる変数を整数型として定義.
110 DEFSNG A,B ──────── A，Bで始まる変数を単精度実数型として定義.
120 DEFDBL D ──────── Dで始まる変数を倍精度実数型として定義.
130 J1=1.23:K=65.643 ──────── J1，Kには整数に変換された値が代入される.
140 ABC=1.23:BBB=65.634
150 D=3.14159265400003#
160 D%=3.14159265400003# ──── D%は，整数型宣言文字(%)が付加されているため，
170 PRINT "J1=";J1,,"K=";K      120行の宣言にかかわらず，整数型として扱われる.
180 PRINT "ABC=";ABC,,"BBB=";BBB
190 PRINT "D=";D,"D%=";D%
200 END
```

## 5．配列変数の宣言と OPTION BASE

```
100 OPTION BASE 1 ―――――――――― 添字の最小値を1とする.
110 DIM KEISAN(9,9) ――――――――― 要素数9×9の2次元配列変数KEISAN()を宣言.
120 FOR I=1 TO 9:FOR J=1 TO 9
130  KEISAN(I,J)=I*J
140 NEXT J,I
150 FOR I=1 TO 9:FOR J=1 TO 9
160  PRINT USING "####";KEISAN(J,I);
170 NEXT J:PRINT
180 NEXT I
190 PRINT:PRINT
200 ERASE KEISAN ――――――――――― KEISAN()を消去.
210 DIM KEISAN(9,9) ――――――――― KEISAN()を新たに宣言.
220 FOR I=1 TO 9:FOR J=1 TO 9
230  KEISAN(I,J)=I+J
240 NEXT J,I
250 FOR I=1 TO 9:FOR J=1 TO 9
260  PRINT USING "####";KEISAN(J,I);
270 NEXT J:PRINT
280 NEXT I
290 END
```

## 6．FOR～NEXT ループ

```
100 SCREEN Ø,Ø,Ø,1:CLS 3
110 COLOR ,,,7
120 FOR P=1 TO 7:C=Ø
130  FOR R=1Ø TO 2ØØ STEP 2Ø
140   FOR I=Ø TO 3.14 STEP .Ø5
150    Y=SIN(I)*COS(I)
160    X=SIN(I)*SIN(I)
170    Y=Y*R+1ØØ
180    X=X*R*2.5+1ØØ-C
190    PSET (X,Y),P
200   NEXT I
210   C=C+1Ø
220 NEXT R,P
230 END
```

― 楕円をプロットする.（140～200）

― 楕円の大きさを変える.（130～220）

― 描き始める位置を変える.（210）

― パレット番号を変える.（120～220）

## 7．サブルーチンの呼び出しとリターン

```
100 *START
110 INPUT "底 辺：";TEIHEN
120 INPUT "高 さ：";TAKASA
130 GOSUB *MENSEKI
140 PRINT "面 積 は   ";MENSEKI
150 PRINT
160 GOTO *START
170 *MENSEKI ――――――――――― 三角形の面積を計算するサブルーチン.
180 MENSEKI=TEIHEN*TAKASA/2
190 RETURN ――――――――――――― サブルーチンの最後はRETURNで終わる.
```

## 8．条件分岐

```
100 INPUT A
110 FLG=Ø
120 IF A<Ø THEN A=ABS(A):FLG=1 ――――― 入力された値が負ならばFLGに1を代入.
130 PRINT SQR(A); ―――――――――――――― 平方根の計算.
140 IF FLG THEN PRINT "i" ELSE PRINT ―― 入力値が負のときには，平方根の結果に
150 END                                  〝i〞（複素表示）を付加する.
```

## 9．式の値による分岐

```
100 *ENTRY
110 INPUT "N=";N
120 SG=SGN(N)+2 ―――――――――― 入力値に対するSGN関数の値－1，0，＋1を，1，2，3に変換．
130 ON SG GOTO *MINUS,*ZERO,*PLUS ―――― 入力値の負，0，正に従って，＊MINUS，
140 *MINUS:PRINT "MINUS!":GOTO *ENTRY        ＊ZERO，＊PLUSに飛ぶ．
150 *ZERO:PRINT "ZERO!":GOTO *ENTRY
160 *PLUS:PRINT "PLUS!":GOTO *ENTRY
```

## 10．変数の値の入れ換え

```
100 DIM A(10),B(10)
110 CLS
120 FOR I=1 TO 10
130  A(I)=I*2:B(I)=I*3:GOSUB *PRINTAB
140 NEXT I:LOCATE 0,Y+3
150 FOR I=1 TO 10
160  SWAP A(I),B(I):GOSUB *PRINTAB ―――― A()とB()の各要素をすべて交換．
170 NEXT I:LOCATE 0,Y+2
180 END
190 *PRINTAB ―――――――――――――――― A()とB()をそれぞれ表示するサブルーチン．
200 X=POS(0):Y=CSRLIN
210 LOCATE X,Y:PRINT USING "###";A(I);
220 LOCATE X,Y+1:PRINT USING "###";B(I);
230 LOCATE X+5,Y
240 RETURN
```

## 11．WHILE～WEND ループ

```
100 I=1
110 WHILE I<=20 ―――――――― Iが20以下のうちは120行～190行を繰り返す．
120  PRINT USING "## -";I;
130  J=1
140   WHILE J<=I ―――――――― Jが I以下のうちは150行～170行を繰り返す．
150    PRINT USING "###";J;
160    J=J+1
170   WEND
180  PRINT:I=I+1
190 WEND
```

## 12．PRINT と WRITE

```
100 A%=123
110 B!=9.87654E+31
120 C#=3.14159265359#
130 D$="N88-BASIC"
140 PRINT A%;B!,C#;D$ ―┐
150 WRITE A%;B!,C#;D$ ―┴― PRINTとWRITEでは出力結果が異なる．
160 END
```

## 13. データの編集出力

```
100 PRINT USING "!";"NEC COMPUTER"
110 PRINT USING "&        &";"NEC COMPUTER"
120 PRINT USING "#####";123.456
130 PRINT USING "#####.#";123.456
140 PRINT USING "+####.#";123.456
150 PRINT USING "#####.#-";-123.456
160 PRINT USING "**###.#";123.456
170 PRINT USING "¥¥###.#";123.456
180 PRINT USING "**¥##.#";123.456
190 PRINT USING "#####,.#";1234.56
200 PRINT USING "#####,.#^^^^";1234.56
210 PRINT USING "¥¥###._-";123.456
220 PRINT USING "#####";123456!
230 PRINT USING "NEC @ COMPUTER";"PERSONAL"
240 END
```

100〜230行 ── 書式制御文字列の典型的な使い方.

## 14. 円の描画

```
100 SCREEN 0,0,0,1:CLS 3
110 FOR J=0 TO 7 :COLOR=(J,J):NEXT J
120 FOR R=1 TO 100 STEP 5
130  CIRCLE(150,100),R,(R MOD 7)+1,,,R/100 ──── 半径, パレット番号, 比率を変えながら楕円を描く.
140 NEXT R
150 ST=.00001
160 DATA 25,5,40,13,17
170 FOR I=1 TO 5
180  READ DAT:EN=ST+DAT/100*3.14*2 ──── DATA行(160行)の百分率データを円の角度に変換.
190  CIRCLE(450,100),100,I,-ST,-EN ──── パレット番号, 開始角度, 終了角度を変えながら扇形を描き, 円グラフを作る.
200  ST=EN
210 NEXT I
```

## 15. パレット(8色中・8色モード)

```
100 SCREEN 0,0,0,1:COLOR,,,,0:CLS 3 ──── 8色中・8色モード. 拡張グラフィックモード使用時のみ.
110 FOR I=1 TO 7
120  COLOR=(I,7) ──── すべてのパレットを白に設定.
130 NEXT I
140 FOR I=1 TO 7
150  LINE(0,I*20)-STEP(639,10),I,BF ──── パレット番号ごとに1本ずつ, 計7本の帯を描く.
160 NEXT I
170 FOR I=1 TO 7
180  FOR J=1 TO 7
190   COLOR=(J,I) ──── 各パレット番号の帯に, 1〜7のカラーコードを順に設定していく.
200   GOSUB *WAITSUB
210  NEXT J
220 NEXT I
230 COLOR ──── 全パレットを初期化.
240 END
250 *WAITSUB ──── 遅延用サブルーチン.
260 FOR K=0 TO 100
270 NEXT K
280 RETURN
```

## 16．パレット(4096色中・16色モード)

```
100 SCREEN 3,0,0,1:COLOR,,,,2:CLS 3 ――― 高分解能カラーモード，4096色中・16色モード．
                                          拡張グラフィックモード使用時のみ．
110 FOR I=1 TO 15
120  COLOR=(I,&HFFF) ――― すべてのパレットを白に設定．
130 NEXT I
140 FOR I=1 TO 15
150  LINE(0,I*20)-STEP(639,10),I,BF ――― パレット番号ごとに1本ずつ，計15本の帯を描く．
160 NEXT I
170 FOR I=1 TO 15
180  FOR J=1 TO 15
190   COLOR=(J,16*I^2) ――― 各パレット番号の帯に，カラーコードを順に設定していく．
200    GOSUB *WAITSUB
210  NEXT J
220 NEXT I
230 COLOR ――― 全パレットを初期化．
240 END
250 *WAITSUB ――― 遅延用サブルーチン．
260 FOR K=0 TO 100
270 NEXT K
280 RETURN
```

## 17．DRAW

```
100 SCREEN 0,0,0,1:CLS 3
110 POINT(320,100)         ┐
120 A$="C4U60R60D60L60"    │
130 DRAW A$                ├― DRAWで図形を描く．
140 DRAW "BE45A2S0.5X=A$;" │
150 DRAW "A0BE10P"         ┘
160 PSET (440,40)          ┐
170 LINE -STEP(60,60),7,BF │
180 PRESET (455,55)        ├― LINE，PAINTで同じ図形を描く．
190 LINE -STEP(30,30),0,B  │
200 PAINT (456,56),0       ┘
210 END
```

## 18．グラフィックパターンの読み込みと表示

```
100 SCREEN 0,0,0,1:CLS 3
110 XD=40:YD=20              ┐
120 BYTE=((XD+7)¥8)*YD*3+3   ├― 8色モードで40×20ドットのパターンを
130 FACT=BYTE¥2+1            │  読み込むための配列変数
140 DIM G%(FACT)             ┘  G%()を確保する．
150 LINE(0,0)-STEP(XD-1,YD-1),1,B
160 CIRCLE(XD/2-1,YD/2-1),YD/2,2
170 GET(0,0)-STEP(XD-1,YD-1),G% ――― 図形をG%()に読み込む．
180 FOR X=0 TO 500 STEP 100
190  PUT(X,100),G% ――― 読み込んだ図形を6か所にPUT．
200 NEXT
```

## 19．グラフィック画面のスクロール

```
100 SCREEN 1,0,0,1:CLS 3
110 FOR I=&H3000 TO &H4F00 STEP &H100
120  FOR J=&H21 TO &H7E
130   KCODE=I+J
140   PUT(X,168),KANJI(KCODE),PSET ――― 漢字をコード順に表示．
150   X=X+20
160   IF X>623 THEN X=0:ROLL 18 ――― 18ドット分ずつスクロールアップ．
170  NEXT J
180 NEXT I
```

## 20. ビューポートの設定

```
100 SCREEN 0,0,0,1:CLS 3
110 WINDOW(0,0)-(639,199) ―――― ウィンドウの設定.
120 C=6:GOSUB *RECT ――┐
130 VIEW(1,1)-(638,99),0,7  │
140 C=5:GOSUB *RECT      ├― ビューポートの設定を変えなが
150 VIEW(214,1)-(428,198),0,7 │   ら，同一座標の長方形を描く.
160 C=4:GOSUB *RECT ――┘
170 VIEW(0,0)-(639,199):CLS 3 ―――― ビューポートをもとに戻す.
180 END
190 *RECT ―――― べた塗りの長方形を描くサブルーチン.
200 LINE(50,50)-(600,150),C,BF
210 LOCATE 0,0:PRINT "どれかキーを押してください";
220 A$=INPUT$(1)
230 RETURN
```

## 21. ウィンドウの設定

```
100 SCREEN 0,0,0,1:CLS 3
110 C=1
120 VIEW(200,50)-(400,150),,7 ―――― ビューポートの設定.
130 WINDOW(-100,-100)-(100,100) ――┐
140  GOSUB *CIRC                  │
150 WINDOW(-100,-100)-(0,0)       │
160  GOSUB *CIRC                  │
170 WINDOW(0,0)-(100,100)         ├― ウィンドウの設定を変えながら，同一座標の円を描く.
180  GOSUB *CIRC                  │
190 WINDOW(-500,-500)-(500,500)   │
200  GOSUB *CIRC ――――――――――――――――┘
210 VIEW(0,0)-(639,199):CLS 3 ―――― ビューポートをもとに戻す.
220 END
230 *CIRC ―――― 円を描くサブルーチン.
240 CIRCLE(0,0),100,C
250 LOCATE 0,0:PRINT "どれかキーを押してください";
260 A$=INPUT$(1)
270 C=C+1:CLS 2
280 RETURN
```

## 22. エラー処理ルーチン

```
100 PRINT "XのY乗を求めます"
110 ON ERROR GOTO *ERRORMES ―――― エラー発生時の分岐先を定義.
120 *START
130 INPUT "X:";X:INPUT"Y:";Y
140 IF X=0 THEN ERROR 250 ―――― エラー番号250を独自に定義.
150 Z=X^Y:PRINT X;"の";Y;"乗は";Z;"です"
160 *RETRY
170 INPUT "もう一度やりますか(Y/N)";A$:PRINT
180 IF A$="Y" OR A$="y" THEN *START
190 ON ERROR GOTO 0 ―――― プログラム終了時には必ず実行しておく.
200 END
210 *ERRORMES ―――― エラー処理ルーチン.
220 IF ERR=250 THEN PRINT "定義されません" ―― エラー番号250のときの処理.
230 IF ERR=6 THEN PRINT "オーバーフローです" ―― オーバーフローのときの処理.
240 RESUME *RETRY ―――― エラー処理ルーチンからの戻りには
                        RESUMEを使う.
```

## 23. 文字とキャラクタコードの変換

```
100 LINE INPUT "英数字を打ってください ";A$
110 IF A$="" THEN 190
120 B$=""
130 FOR I=1 TO LEN(A$)
140  D$=MID$(A$,I,1):D=ASC(D$)
150  IF D>=97 AND D<=122 THEN D$=CHR$(D-32) ──── 英小文字のみを，英大文字に変換．
160  B$=B$+D$
170 NEXT I
180 PRINT:PRINT B$
190 END
```

## 24. 8進変換と16進変換

```
100 FOR I=0 TO 16
110  DE$=RIGHT$("  "+STR$(I),3) ─┐
120  OC$=RIGHT$("  "+OCT$(I),3) ─┼─ 0～16の整数を，10進表記，8進表記，
130  HE$=RIGHT$("  "+HEX$(I),3) ─┘   16進表記の文字列に変換．
140  PRINT "10進：";DE$,
150  PRINT " 8進：";OC$,
160  PRINT "16進：";HE$
170 NEXT I
```

## 25. 文字列の抜き出し

```
100 INPUT"英数字をなにか打ってください ",A$
110 PRINT
120 PRINT "全体を２つに分割します"
130 AL$=LEFT$(A$,LEN(A$)¥2) ──────────────┐
140 AR$=RIGHT$(A$,LEN(A$)-LEN(A$)¥2) ─────┴─ A$を左右半分ずつに分割する．
150 PRINT AL$;SPACE$(3);AR$
160 PRINT
170 PRINT "２文字ずつに分けます"
180 FOR I=1 TO LEN(A$) STEP 2 ──────────┐
190  PRINT MID$(A$,I,2);SPACE$(3);       ├─ A$を2文字ずつに分ける．
200 NEXT ───────────────────────────────┘
210 END
```

## 26. 数値データの文字型化

```
100 OPEN "RDATA" AS #1 ──────────────── ランダムモードで〝RDATA〟をオープン．
110 FIELD #1,2 AS A$,4 AS B$,8 AS C$ ── 整数用2バイト，単精度実数用4バイト，倍精度
120 A%=987:A!=123.456:A#=1234567891234#   実数用8バイトのフィールド変数を定義．
130 LSET A$=MKI$(A%) ─┐
140 LSET B$=MKS$(A!)  ├─ 各型の数値を文字型データに変換し，
150 LSET C$=MKD$(A#) ─┘   フィールド変数にセット．
160 PUT #1 ──────────────────────────── ファイルに書き込む．
170 PRINT "このプログラムで書き込むデータは"
180 PRINT "次のプログラムで読み出せます"
190 CLOSE:END
```

## 27. 文字型化数値データの数値化

```
100 OPEN "RDATA" AS #1 ──────────────── ランダムモードで〝RDATA〟をオープン．
110 FIELD #1,2 AS A$,4 AS B$,8 AS C$ ── 整数用2バイト，単精度実数用4バイト，
120 PRINT "前のプログラムで書き込んだデータを"  倍精度実数用8バイトのフィールド変数
130 PRINT "読み出します"                        を定義．
140 GET #1 ──────────────────────────── ファイルから読み込む．
150 A%=CVI(A$) ─┐
160 A!=CVS(B$)  ├─ 文字型データを，各型の数値に戻す．
170 A#=CVD(C$) ─┘
180 PRINT A%:PRINT A!:PRINT A#
190 CLOSE:END
```

## 28. シーケンシャルファイル

```
100 F$="DATA.D"+RIGHT$(DATE$,2)
110 OPEN F$ FOR OUTPUT AS #1 ──── “DATA.Dxx”(xxは日付)を出力モードでオープン.
120 PRINT #1,DATE$;",";TIME$ ──── 日付と時刻を書き出す.
130 PRINT "今日のデータを入力してください"
140 PRINT:PRINT "リターンキーだけ押すと終了します"
150 PRINT:INPUT "品名";NA$
160 IF NA$="" THEN CLOSE:GOTO *INDATA ──── 入力の終了．ファイルのクローズ.
170 INPUT "価格";PRC
180 INPUT "個数";N%
190 PRINT #1,NA$;",";PRC;",";N% ──── 品名，価格，個数の各データを書き出す.
200 GOTO 140
210 *INDATA
220 TOTAL=0
230 OPEN F$ FOR INPUT AS #1 ──── 同じファイルを入力モードでオープン.
240 INPUT #1,DA$,TI$ ──── 日付と時刻を読み込む.
250 PRINT:PRINT "日付    :  ";DA$,"時刻    :  ";TI$
260 PRINT
270 IF EOF(1) THEN *TOTALPRN
280 INPUT #1,NA$,PRC,N% ──── 品名，価格，個数の各データを読み込む.
290 SUM=PRC*N% ──── 小計.
300 PRINT NA$;TAB(10);PRC;"*";N%;"=";SUM ──── 品名，価格，個数および小計の表示.
310 TOTAL=TOTAL+SUM ──── 合計.
320 GOTO 270
330 *TOTALPRN
340 PRINT:PRINT "TOTAL=";TOTAL ──── 合計の表示.
350 PRINT:PRINT "今日のデータを ";F$;" として作成しました"
360 CLOSE:END ──── ファイルをクローズ，終了.
```

## 29. ランダムファイル

```
100 CLS:PRINT"住所録のファイル名は？"
110 PRINT:LINE INPUT F$
120 IF F$="" THEN 100
130 OPEN F$ AS #1 ──── 指定ファイル名のファイルをランダムモードでオープン.
140 FIELD #1,30 AS NAM$,20 AS TEL$,50 AS ADR$ ──── 名前(30バイト)，TEL(20バイト)，住所(50バイト)の専用フィールド変数を定義.
150 *MENU
160 PRINT:PRINT
170 PRINT "<<メニュー>>":PRINT
180 PRINT "  検索／修正 : 1"
190 PRINT "  新しく登録 : 2"
200 PRINT "  終  わ  り : 0"
210 PRINT:PRINT "操作を番号で選択してください ";
220 Q$=INPUT$(1):PRINT Q$
230 ON VAL(Q$)+1 GOTO *EXIT,*INNAME,*NEWENTRY
240 GOTO *MENU
250 *INNAME ──── 検索のルーチン.
260 IF LOF(1)<>0 THEN 330 ─┐
270 PRINT:PRINT F$+" にはデータがありません"   │
280 PRINT:PRINT"新しく登録しますか？(Y/N) ";   │
290 Q$=INKEY$:IF Q$="" THEN GOTO 290          ├── ファイルにまったくデータがない場合の例外処理.
300 IF Q$="N" OR Q$="n" THEN GOTO *MENU       │
310 IF Q$="Y" OR Q$="y" THEN GOTO *NEWENTRY   │
320 GOTO 290 ─┘
330 NH=1
340 PRINT:PRINT "捜す名前は？":PRINT:INPUT "名前:";HNAM$ ── 検索したい名前の入力.
350 IF HNAM$="" THEN *MENU
360 HL=LEN(HNAM$)
370 IF NH>LOF(1) THEN PRINT:PRINT"その名前は登録されていません":GOTO *MENU ─┐
380 GET #1,NH                                                               ├── ファイル中から1レコードずつ読み込み，名前を検索する.
390 IF HNAM$<>LEFT$(NAM$,HL) THEN NH=NH+1:GOTO 370 ─┘
400 PRINT:PRINT"名前:";NAM$ ─┐
410 PRINT"TEL:";TEL$          ├── 検索結果の表示.
420 PRINT"住所:";ADR$ ─┘
430 PRINT:PRINT"修正しますか？(Y/N) "; ─┐
440 Q$=INKEY$:IF Q$="" THEN GOTO 440            │
450 IF Q$="N" OR Q$="n" THEN PRINT "N":GOTO *MENU   ├── 修正の要，不要を決める.
460 IF Q$="Y" OR Q$="y" THEN PRINT "Y":GOTO *CHANGE │
470 GOTO 440 ─┘
```

```
480 *CHANGE ────────────────────────────────── 修正のルーチン.
490 PRINT:PRINT"リターンキーだけ押すとその項目は修正されません"
500 DNAM$="":DTEL$="":DADR$="" ─┐
510 PRINT:INPUT"名前";DNAM$     │
520 IF DNAM$<>"" THEN LSET NAM$=DNAM$ │
530 INPUT"TEL";DTEL$            ├── 各データを各フィールド変数にセット.
540 IF DTEL$<>"" THEN LSET TEL$=DTEL$ │
550 INPUT"住所";DADR$           │
560 IF DADR$<>"" THEN LSET ADR$=DADR$ ─┘
570 PUT #1,NH:GOTO *MENU ─────────── 修正後のデータを書き出す.
580 *NEWENTRY ────────────────────── 新規登録のルーチン.
590 PRINT:PRINT "新しく登録します"
600 NH=LOF(1)+1 ─────────────────── ファイルの最終レコードの次のレコードを指定.
610 PRINT:INPUT"名前:";NNAM$ ─┐
620 IF NNAM$="" THEN *MENU     │
630 INPUT"TEL:";NTEL$          │
640 INPUT"住所:";NADR$         ├── 各データを各フィールド変数にセット.
650 LSET NAM$=NNAM$            │
660 LSET TEL$=NTEL$            │
670 LSET ADR$=NADR$ ───────────┘
680 PUT #1,NH:GOTO *MENU ─────────── データを書き出す.
690 *EXIT ────────────────────────── 終了のルーチン.
700 PRINT:PRINT"終了"
710 CLOSE:END ────────────────────── ファイルのクローズ, 終了.
```

## 30. キー割り込みルーチン

```
100 CLS
110 ON KEY GOSUB *SINSUB,*COSSUB,*ENDSUB ── f・1, f・2, f・3が押されたときの分岐先を定義.
120 FOR I=1 TO 3
130  KEY(I) ON ───────────────────── f・1, f・2, f・3の各キーの割り込みを許可.
140 NEXT I
150 DEG=0:PI=3.1416:FLAG=1
160 PRINT "f・1 または f・2 を押してください (f・3 で終了)."
170 *START
180 TH=DEG/180*PI
190 LOCATE 0,10
200 ON FLAG GOTO *SINC,*COSC                                      10度ごとの
210 *SINC:PRINT "TH=";DEG;TAB(12);"SIN(TH)=";SIN(TH);:GOTO *EXIT ── 正弦値を表示.
220 *COSC:PRINT "TH=";DEG;TAB(35);"COS(TH)=";COS(TH);:GOTO *EXIT ── 10度ごとの
230 *EXIT:DEG=DEG+10                                              余弦値を表示.
240 FOR I=0 TO 1000:NEXT ─────────── 表示遅延用タイマー.
250 LOCATE 0,10:PRINT STRING$(70," ");
260 GOTO *START
270 *SINSUB:FLAG=1:RETURN ─────────── f・1が押されたらFLAG=1にする.
280 *COSSUB:FLAG=2:RETURN ─────────── f・2が押されたらFLAG=2にする.
290 *ENDSUB:KEY OFF:END ───────────── f・3が押されたら終了.
```

## 31. タイマー割り込みルーチン

```
100 CLS:CONSOLE ,,,1:FLG=0
110 PRINT "目覚まし時計"
120 LOCATE 0,10:PRINT "現在の時刻は ";TIME$
130 LOCATE 0,2:INPUT "セット時刻は (HH:MM:SS) ";ST$
140 ON TIME$=ST$ GOSUB 200 ────────── セット時刻到来時の分岐先を定義.
150 TIME$ ON ──────────────────────── クロック割り込みを許可.
160 LOCATE 13,10:PRINT TIME$; ─────── 現在時刻の表示. セット時刻までは
170 IF FLG=0 THEN 160                 160行～170行が繰り返し実行される.
180 CLS:LOCATE 30,10:PRINT "おはよう!!!"
190 END
200 PRINT                             なにかキーを押さない限り, メッ
210 COLOR 7:PRINT "もう起きる時間ですよ!!!"; セージが表示され, スピーカは鳴
220 IF INKEY$="" THEN BEEP 1:GOTO 210 ── りっぱなしとなる.
230 BEEP 0:FLG=1 ──────────────────── なにかキーが押されると, スピーカは
240 RETURN                            鳴りやみ, メインルーチンに戻る.
```

## 32. 乱数の発生

```
100 RANDOMIZE
110 DIM SUM(6)
120 FOR I=1 TO 100
130  DA=INT(RND*6+1)
140  SUM(DA)=SUM(DA)+1
150 NEXT I
160 FOR I=1 TO 6
170  PRINT I;"-";SUM(I);"%"
180 NEXT I
```

## 33. テキスト画面のモード設定と文字に対する色・機能の設定

```
100 WIDTH 80,25:CLS
110 CONSOLE 0,25,0,0 ──────── 白黒モード.
120 LOCATE 0,0
130 PRINT "白黒モード":PRINT
140 GOSUB *PRINTCHR
150 GOSUB *KEYSUB
160 CONSOLE 0,25,0,1 ──────── カラーモード.
170 LOCATE 0,0
180 PRINT "カラーモード":PRINT
190 GOSUB *PRINTCHR
200 END
210 *PRINTCHR
220 FOR I=0 TO 31
230  PRINT "-COMPUTER-";
240 NEXT I
250 GOSUB *KEYSUB
260 FOR I=0 TO 7
270  COLOR@(I*10,2)-(I*10+9,5),I ── 文字にファンクションコードを設定.
280 NEXT I
290 RETURN
300 *KEYSUB ──────── キー入力待ちサブルーチン.
310 LOCATE 0,10
320 PRINT "どれかキーを押してください";
330 A$=INPUT$(1)
340 RETURN
```

## 34. 利用者定義文字の登録

```
100 DIM CHRPTN%(17) ──────── 利用者定義文字パターンを格納するための配列変数を確保.
110 CHRPTN%(0)=16:CHRPTN%(1)=16
120 FOR I=2 TO 17:CHRPTN%(I)=2^(I-2)-1:NEXT I ──── パターンを格納する.
130 KPLOAD &H762A,CHRPTN% ──────── パターンをシステムに登録.
140 FOR I=0 TO 4
150  PRINT KNJ$("762A"); ──────── 表示.
160 NEXT
```

## 35. 2バイト系日本語文字列の変換と抜き出し

```
100 A0$="ｲﾛﾊ漢字ABC"
110 PRINT A0$
120 K0$=AKCNV$(A0$) ──────── 1バイト系英数カナ文字を2バイト系全角文字に変換.
130 PRINT K0$
140 K1$=KMID$(K0$,1,3)+KMID$(K0$,6,3) ──── 2バイト系全角文字に変換された文字列から
                                            "漢字"を取り除く.
150 PRINT K1$
160 A1$=KACNV$(K1$) ──────── 2バイト系全角文字を1バイト系英数カナ文字に
                              変換.
170 PRINT A1$
```

## 36. 2バイト系日本語文字列と漢字コードの相互変換

```
100 A$=JIS$(KMID$("漢字",1,1)) ─── "漢"のJISコードを求める.
110 PRINT A$
120 B$=KNJ$(A$) ─── "漢"を表示する.
130 PRINT B$
```

## 37. 特定タイプの文字列の抜き出し

```
100 A$="ｲﾛﾊ漢字ABC"
110 PRINT A$
120 PRINT KEXT$(A$,0) ─── 1バイト系英数カナ文字のみを抜き出して表示.
130 PRINT KEXT$(A$,1) ─── 2バイト系日本語文字のみを抜き出して表示.
```

## 38. 2バイト系日本語文字を含む文字列の長さとタイプ

```
100 A$="ABC漢字ｲﾛﾊ"
110 PRINT A$:PRINT
120 FOR I=0 TO 4
130  PRINT KLEN(A$,I); ─── 各タイプの文字の合計数を，タイプ別に表示.
140 NEXT I
150 PRINT:PRINT
160 FOR J=1 TO KLEN(A$,0) ─── 文字列中の各文字のタイプを順に表示.
170  PRINT KTYPE(A$,J);
180 NEXT J
```

## 39. 利用者定義関数

```
100 DEF FNMENSEKI(R)=3.14159*R*R ─── 円の面積を求める利用者定義関数の定義.
110 DEF FNENSHUU(R)=3.14159*2*R ─── 円周を求める利用者定義関数の定義.
120 INPUT "半径は";HANKEI
130 PRINT "面積:";FNMENSEKI(HANKEI)
140 PRINT "円周:";FNENSHUU(HANKEI)
150 END
```

## 40. 電話制御機能(拡張機能)

```
100 CMD LINE OPEN "COM1:" AS #1 ─── 電話機をコミュニケーションポート1に論理的に接続することを宣言.
110 CMD ON LINE #1 GOSUB *IL ─── 着信割り込みルーチンの設定.
120 CMD LINE #1 ON ─── 着信割り込みの許可.
130 GOTO 130 ─── 着信を待機する.
140 *IL
150 CMD RECEIVE #1,1 ─── 着信時，データ通信モードに入る.
160 WHILE STATUS MODE(#1)<>2:WEND ─── 完全にデータ通信モードになるまで待つ.
170 OPEN "COM1:E71N" AS #1 ─── プロトコル設定.
180 CMD ON ERROR #1 GOSUB *IE ─── 通信エラー割り込みルーチンの設定.
190 CMD ERROR #1 ON
200 FLN$="":A$=""
210 FOR I=1 TO 3000:NEXT
220 PRINT #1,CHR$(&H15); ─── キャラクタコード&H15を送信，通信を開始.
230 IF LOC(1)=0 THEN 230 ─┐
240 A$=INPUT$(1,1)         │
250 IF A$=CHR$(&HD) THEN 230 │
260 IF A$=CHR$(&HA) THEN 290 ├── 受信した最初の1行をFLN$に代入.
270 FLN$=FLN$+A$           │
280 GOTO 230 ──────────────┘
290 IF FLN$="" THEN 360
300 OPEN FLN$ FOR OUTPUT AS #2 ─── 受信ファイルをFLN$に代入された名前でオープン.
310 IF LOC(1)=0 THEN 310
320 A$=INPUT$(1,1)
330 IF A$=CHR$(&H4) THEN 360 ─── キャラクタコード&H4を受信，通信を終了.
340 PRINT #2,A$; ─── 受信データをファイルに書き込む.
350 GOTO 310
```

```
360 CLOSE
370 CMD MODE CUT #1 ――――――――――― 電話を切る.
380 CMD RECEIVE #1,0 ――――――――― 自動着信を禁止.
390 RETURN
400 *IE ―――――――――――――――――― 通信エラー割り込みルーチン.
410 PRINT "通信エラー発生":BEEP
420 DMY$=INPUT$(LOC(1),1) ―――――― 通信エラー情報のダミー読み取り.
430 RETURN 360                    エラー状態はリセットされる.
```

## 41. サウンド機能(拡張機能)

```
100 CLEAR &H100:I=0
110 GOSUB *BEGIN :GOSUB *MPLAY
120 ON PLAY (3,0) GOSUB *MPLAY ― チャンネル番号3のサウンドバッファ内の未演奏音楽情報が0以下
130 PLAY ON                        になったときに*PLAYに飛ぶように宣言.
140   X=RND*639:Y=RND*399:C=(RND*100 MOD 7)+1 ┐
150   CIRCLE (X,Y),RND*20,C                    ├― 140〜170行の間でPLAY割り込み
160   PAINT (X,Y),(RND*100 MOD 7)+1,C          │   を待っている.
170 GOTO 140 ――――――――――――――――――――――――――――――┘
180 REM
190 *MPLAY ―――――――――――― 演奏ルーチン.
200 PLAY STOP:CLS 3
210 PLAY A$,B$,B$ ┐
220 PLAY B$,A$,B$ ├― 演奏.
230 PLAY B$,B$,A$ ┘
240 PLAY ON
250 RETURN
260 *BEGIN ―――――――――――― 初期化ルーチン.
270 CLS 3
280 CONSOLE 0,25,0,1
290 SCREEN 3,0                        サウンドバッファの確保及び初期化. チャンネル1〜3のサウン
300 PLAY ALLOC 255,255,255 ―――――― ドバッファには255バイトが割り当てられ, 他のサウンドバッフ
310 INI$="MB@55@V100"                 ァは0バイトとなる.
320 PLAY INI$,INI$,INI$ ―――――――― チャンネル1〜3の初期化設定を行う.
330 A1$="T160O4L8C4CC4DE4EE4ED4CD4EC4CO3G4RO4E4." ――――――――┐
340 A2$="EEFG4GG4GF4EF4GE4.R4GE2RGE2RGER8GER8GE4.R4" ―――┴― 演奏データ.
350 A$=A1$+A2$
360 B$="R2R2R2R2R2R2"
370 I=I+1
380 RETURN
```

## 42. 10進演算機能(拡張機能)

```
100 CLS
110 DEFSTR A-Z
120 C0.5=PCNV$("0.5"):C1=PCNV$("1") ――――――┐
130 C2=PCNV$("2"):C100=PCNV$("100") ――――――┴―― 定数をパック10進数に変換.
140 INPUT "元金(円)";G:IF NOT PCHK(G) THEN 140 ┐ 入力値の妥当性チェック. 不適当
150 INPUT "年利(%)";R:IF NOT PCHK(R) THEN 150 ┘ な場合は再入力を要求.
160 G=PCNV$(G):R=PCAL$(PCNV$(R),/,C100) ――――――― 変数値(文字列)をパック10進数に
170 FOR NEN!=.5 TO 10 STEP .5                      変換.
180   Y=PCNV$(NEN!,2) ―――――――――――――――――――――― 変数値(単精度)をパック10進数に
190   TMP1=PCAL$(C1,+,PCAL$(R,/,C2)) ――――――┐     変換.
200   TMP2=PCAL$(INT,PCAL$(Y,/,C0.5),1)     ├―― 元利合計の計算.
210   T=PCAL$(G,*,PCAL$(TMP1,^,TMP2)) ―――――┘
220   PRINT "年=";NEN!,"合計=";PCNV$(T,1) ―――――― 演算結果をパック10進数から文字
230 NEXT                                            列に変換して表示.
240 END
```

# 付　録

# A. エラーメッセージとその対策

BASICの実行中にエラーが発生すると，メッセージが表示されます．ここでは，こうしたエラーメッセージと，その原因および対策について説明します．

メッセージの右側の数字(85など)は，〝エラーコード〞と呼ばれるもので，エラー発生時，ERR(関数)に格納されます．

〝原因：〞には，そのエラーが発生した際の原因を簡略に説明してあります．複数の原因が考えられる場合は，列挙してあります．

〝対策：〞には，そのエラーの原因をとり除く方法か，またはエラーの発生を未然に防ぐための対策が説明してあります．

エラーについての詳細は，第2章中のERL／ERR，ERROR，ON　ERROR　GOTO，RESUMEなどの項をご覧ください．

**Access　denied　　85**

原因：ファイルは保護されている．

対策：ファイルの保護状態を解除するか，保護されているファイルに書き込まないようにする．

**Bad　drive　specification　　70**

原因：LOAD, SAVE, KILL, OPENなどのファイル操作命令でドライブの指定がまちがっている．

　　システムにつながっていないドライブを指定した．

対策：正しいドライブを指定する．

**Bad　file　name　　56**

原因：LOAD, SAVE, KILL, OPENなどのファイル操作命令で，ファイル名の指定がまちがっている．

対策：正しいファイル名を指定する．

**Bad　file　number　　52**

原因：同時オープンファイル数を超えるファイル番号を指定しようとした．

対策：ファイル番号を小さくするか，同時オープンファイル数（インタプリタの場合は起動時に指定．コンパイラの場合はREM　$FILEで指定）を増やす．

Bad　telephone　number　　　　　　　173

原因：拡張電話機能の命令／関数で電話番号の指定がまちがっている．
　　　あるいは電話番号として指定できない文字が指定されている．

対策：正しい電話番号を指定する．

Bad　telephone　set　number　　　　　　　172

原因：拡張電話機能の命令／関数で電話機番号の指定がまちがっている．

対策：正しい電話機番号（1，2，3）を指定する．

Can't　continue　　　　　　　17

原因：STOP 命令や CTRL＋C などで実行を停止した場合，プログラムを書き換えたりすると，CONT によって実行を再開することができない．

対策：CONT を使用したい場合は，実行の中断中にプログラムを書き換えてはならない．実行中断のタイミングによっては，プログラムを書き換えたりしなくとも実行の再開ができなくなることもあるので注意．

Can't　execute　across　mode　　　　　　　176

原因：現在のモードでは実行できない命令／関数を実行しようとした．

対策：その命令／関数の実行に有効なモードに再設定する．

CHILD　can't　execute　　　　　　　77

原因：COMMAND．COM がルートディレクトリにないため CHILD が実行できない．

対策：BASIC を起動する前に COMMAND.COM をルートディレクトリにコピーする．

Communication　I／O　error　　　　　　　174

原因：拡張電話機能使用時に電話機の故障，電話機の初期設定の不良，接続不良などにより，電話機に対するコマンドレベルの入出力ができない．

対策：電話機，電話機の初期設定，接続などが不適切でないかどうかチェックする．

Direct　statement　in　file　　　　　　　57

原因：LOAD でアスキー形式のプログラムファイルをロードするときに，ダイレクトステートメント（行番号のない行）が存在した．

対策：プログラムファイルの行番号が何かの理由で破壊されているか，あるいは，指定ファイルがプログラムファイルではないデータファイルであると考えられるので，ファイルの内容を確認する．

**Disk full　　　　　　　68**

**原因**：SAVE, PRINT #, PUT などのディスクに書き出す命令を実行したとき，ディスク上に空き領域がなかった．

**対策**：不要なファイルを削除して，空き領域を確保する．
別のドライブ上のディスクを使用する．
同じドライブ上でディスクを差し替えて使用する．ただしこの方法は，ファイルがオープン状態のときには使えないので注意．

**Disk I／O error　　　　　　　64**

**原因**：ディスクとの入出力中にエラーが発生した．

**対策**：入出力を行おうとしたディスクがフォーマット（MS-DOS による）されていないか，または物理的に破壊されていると考えられるので，ディスクの再フォーマットを行ってみる．

**Disk offline　　　　　　　62**

**原因**：LOAD, SAVE, KILL, OPEN などのファイル操作命令で，ディスクのセットされていないドライブに対して入出力を行おうとした．

**対策**：ディスクを正しくドライブにセットする．

**Division by Zero（／0）　　　　　　　11**

**原因**：0 による除算，すなわち n／0（n は任意の数）や 0^−1 が実行されて，その結果得られた値がオーバーフローを起こした．

**対策**：除数が 0 にならないようにする．

**Duplicate Difinition　　　　　　　10**

**原因**：一度宣言した配列を二重定義しようとした．
DIM で宣言しないで配列を使った場合に，その配列を再定義しようとした．

**対策**：別の配列名を使うか，あるいは，ERASE で古い配列を消去した上で，再び同じ配列名で定義しなおす．

**Duplicate label　　　　　　　31**

**原因**：プログラム中に同じラベル名が 2 つ以上存在している．

**対策**：ラベル名を変更して同一名のラベル名が存在しないようにする．

**Feature not available　　　　　　　33**

**原因**：現在の状態では使用できない命令を実行しようとした．

**対策**：その命令を実行するために必要なハードウェアなどを装備あるいは準備する．

FIELD overflow 50

原因：FIELDで，ランダムファイルのレコード長として合計でレコードサイズ(インタプリタでは起動時に指定．コンパイラではREM $FILEで指定) 以上の領域を指定した．

対策：フィールドの合計長は指定サイズを超えないようにする．

File already exist 65

原因：NAMEによって新たに付けようとしたファイル名がすでに存在している．

対策：新しく付けるファイル名を変更するか，すでに存在しているファイルのファイル名を変更する．

File already open 54

原因：すでにオープンされているファイルに対して，OPEN，KILL，NAMEなどを行った．

対策：CLOSEでいったんファイルを閉じる．

File not found 53

原因：LOAD，SAVE，KILL，NAMEなどのファイル操作命令で，指定したファイルが見つからない．

対策：正しいドライブ名およびファイル名を指定する．

File not open 60

原因：PRINT #，INPUT #など，ファイル番号を指定して入出力を行う命令で，オープンされていないファイルを参照しようとした．

対策：入出力命令を実行する前に，OPENでファイルをオープンしておく．

File write protected 61

原因：次のうちいずれかの書き込み禁止属性が付けられているファイルに，書き込みを行おうとした．

- SETによって，ファイルディスクリプタ，ドライブのディスク，ファイル番号に付けられたもの．
- フロッピィディスクの書き込み禁止ノッチまたはスイッチによるもの．

対策：書き込み禁止を解除する．

FOR without NEXT 26

原因：FOR～NEXTが正しく対応していない（FORが多い）．

対策：プログラムの流れをたどり，FOR～NEXTを正しく対応させる．

Illegal direct 12

原因：ダイレクトモードで使用できない命令をダイレクトモードで使った．

対策：プログラムモード上で実行する．

### Illegal function call 5

原因：命令や関数の使い方がまちがっている．すなわち，引数がその関数の許容する範囲を超えたり，結果がその関数のとり得る範囲を超えたりしている．たとえば，次のような場合である．

- LOG関数で負や0の引数を指定した．
- SQR関数で負の引数を指定した．
- MID$, LEFT$, RIGHT$, STRING$, SPACE$, INSTR, ON…GOSUBなどにおいて，不適当な引数が用いられた．
- ほか．

対策：原因がさまざま考えられるので，各命令，関数の使い方をリファレンスで確認する．

### Illegal operation 74

原因：スクロールウィンドウ内に表示データがいっぱいに詰まっている場合，またはスクロールウィンドウ内でキー入力したデータがいっぱいに詰まっている場合に，スクロール開始行に対してインサートモードでキー入力した．

対策：インサートモードを抜けてから，キー入力を行う．

### Input past end 55

原因：INPUT #, GETなどのファイルからデータを読み込む命令で，ファイル中のすべてのデータを読み尽くしたあとに，さらに入力命令が実行された．

対策：ファイル中のデータの数と，入力命令で読み込む数とを合わせる．

EOF, LOFなどの関数を使って，ファイルの終了を検出するようにする．

### Line buffer overflow 23

原因：1行で入力できる文字の範囲（255バイト）を超えて入力が行われた．

対策：1行の文字数を255バイト以内にする．

### Line busy 175

原因：拡張電話機能使用時に電話の呼び出し相手が話し中である．

対策：再度電話をかける．

### Missing operand 22

原因：命令中で，必要なパラメータが指定されていない．

対策：書式を確かめ，必要なパラメータを指定する．

### Mouse board not ready 78

原因：オプションのマウスインタフェースボードがセッティングされていない（マウスインタフェース標準実装機種以外）．

対策：マウスインタフェースボードの実装状態をチェックする．

### MOUSE not initialized 79

原因：MOUSE 0が実行されていない．

対策：MOUSE 0を実行してマウスを初期化してから，マウス関連の命令を実行するようにする．

### NEXT without FOR 1

原因：FOR〜NEXTが正しく対応していない（NEXTが多い）．

対策：プログラムの流れをたどり，FOR〜NEXTを正しく対応させる．

### No RESUME 19

原因：エラー処理ルーチンの終わりにRESUMEがなく，プログラムの実行が継続できない．

対策：RESUME，END，ON ERROR GOTO 0のどれかで終らせるようにする．

### Out of DATA 4

原因：DATA行中のデータが足りず，READで読むべきデータがない．

対策：DATA行中のデータの数と，READで読み込む数とを合わせる．READとDATAの対応と，RESTOREが正しく使われているかどうかを確認する．

### Out of memory 7

原因：次のような理由により，メモリ容量が足りなくなった．

- ・プログラムが長すぎる．
- ・配列が大きすぎる．
- ・FOR〜NEXT，GOSUBなどでネスティング（入れ子構造）が深くなりすぎて，スタックがいっぱいになった．
- ・ほか．

対策：プログラム，配列を小さくする．ネスティングを浅くする．

FREを使ってフリーなメモリを調べてみて，メモリの空き容量が異常に少ない場合には，そのプログラムを実行する前に別のプログラム中で，BASICのプログラム領域を小さく設定したままにしている可能性がある．このようなときには，CLEARを実行して，BASICの使うメモリを大きくする．また，起動時にファイルバッファの容量を多く取っていることもあるので，再起動して同時オープンファイル数を必要最小限にする．

### Out of string space 14

原因：文字列の全体量が，使用できるメモリ容量を超えてしまった．

対策：文字列，配列文字列を小さくする．

**Overflow（OV）**　6

原因：演算結果や入力された数値が，許される範囲を超えた．

対策：データの型と，とる値の範囲が正しいかどうかを確認する．

**Path／File　access　error**　75

原因：指定したディレクトリ名がすでに存在している．

対策：正しいディレクトリ名を指定する．

**Path　not　fonud**　76

原因：指定したパス（ディレクトリ）名が存在しない．

対策：正しいパス（ディレクトリ）名を指定する．

**Rename　across　disks**　73

原因：異なるドライブ上にあるファイルに対してNAMEを実行しようとした．

対策：NAMEでは同じドライブ上のファイルを指定する．

**Request　denied**　177

原因：拡張電話機能使用時に電話機に対してコマンドを送出したが，電話機にそのコマンドを拒否された．

対策：電話機に対して正しいコマンドを送出する．

**RESUME　without　error**　20

原因：エラー処理ルーチンでないところに，RESUMEがある．

対策：ON　ERROR　GOTOで指定されたところ以外，たとえばメインルーチン内にRESUMEがあるとこのエラーが発生するので，メインルーチンの終わりには必ずENDを置くようにする．プログラムの流れを確認すること．

**RETURN　without　GOSUB**　3

原因：GOSUB～RETURNが正しく対応していない（RETURNが多い）．

対策：GOSUBで呼び出されたサブルーチン以外，たとえばメインルーチン内にRETURNがあるとこのエラーが発生するので，プログラムの流れに沿って，GOSUBとRETURNの対応を確認する．

**RS-232C（2nd／3rd）board　not　ready**　82

原因：RS-232C拡張ボードが接続（実装）されていないのに，電話機と標準以外のRS-232C回線をCMD　LINE　OPENで接続しようとした．

対策：RS-232C拡張ボードを接続（実装）するか，標準のRS-232C回線と電話機とを接続する．

## Sequential I/O only 59

**原因**：シーケンシャル入出力以外の入出力命令を使用した.
MERGEでバイナリファイルを指定した.

**対策**：シーケンシャル入出力命令を使用するようにする.
MERGEでディスクから読み込むファイルは，アスキー形式でセーブしておく.

## Sharing violation 84

**原因**：すでに他プロセスが共用モード指定なしでファイルをオープンしているか，あるいはすでに自分自身が共用モード指定つきでファイルをオープンしているのにそのファイルを共用モード指定なしでオープンしようとした.

**対策**：共用モード指定つきでオープンする．あるいは先に自分自身がオープンしているファイルをクローズしてから命令を実行する.

## String formula too complex 16

**原因**：文字式が複雑すぎる.

**対策**：複雑な文字式は，いくつかの式に分ける.

## String too long 15

**原因**：1つの文字変数内の文字数が255バイトを超えている.

**対策**：2つの変数に分けて処理する.

## Subscript out of range 9

**原因**：配列変数の添字の値がDIMおよびOPTION BASEで指定されたときの範囲を外れている.
DIMによる宣言を省略した場合に，添字の値が10を超えている.

**対策**：配列変数の定義による添字の範囲と引用する配列変数の範囲を確認する.

## Syntax error 2

**原因**：命令が書式どおりになっていない.
予約語以外の命令がある.
READとDATAとの対応で，読み込む変数とデータとの間で型の不一致が生じた.

**対策**：各命令を書式どおり正しく記述する.
READとDATAでは，読み込む変数とデータの型を合わせる.

## Telephone set not open 170

**原因**：拡張電話機能使用時に電話機が論理的に接続されていない.

**対策**：CMD LINE OPENを実行してから，他の拡張電話制御命令を実行するようにする.

### Telephone　set　already　open　　　171

原因：拡張電話機能使用時に，すでにCMD　LINE　OPENが実行された電話機に対して，再度この命令を実行した．

対策：CMD　LINE　OPENが重複しないようにする．

### Type　mismatch　　　13

原因：式の左辺，右辺，関数の引数などにおいて，変数の型が一致していない．

対策：関数に与えるデータの型を確認する．

### Undefined　label　　　32

原因：定義されていないラベル名を参照した．

対策：ラベル名を正しく定義する．

### Undefined　line　number　　　8

原因：GOTO, GOSUB, IF　THEN　ELSEなどで，存在しない行番号を飛び先に指定した．

対策：行番号を確認する．

### Undefined　user　function　　　18

原因：定義されていない利用者定義関数，または機械語関数を引用した．

対策：利用者定義関数は，引用する前にDEF　FNで定義する．機械語関数は，DEF　USRで実行開始アドレスを定義しておく．

### Unprintable　error　　　21，24，25，28，他

原因：メッセージの定義されていないエラー．

対策：ERROR　〈整数表記〉を実行してこのエラーが生ずる場合，その〈整数表記〉番のエラーをユーザー独自のエラー処理に利用することができる．

### WEND　without　WHILE　　　30

原因：WHILE～WENDが正しく対応していない（WENDが多い）．

対策：プログラムの流れをたどり，WHILE～WENDを正しく対応させる．

### WHILE　without　WEND　　　29

原因：WHILE～WENDが正しく対応していない（WHILEが多い）．

対策：プログラムの流れをたどり，WHILE～WENDを正しく対応させる．

### システム予約　　　51

原因：BASICで予約されたエラー番号．

対策：利用者が使うことはできない．

# B. コントロールコード表

| キャラクタコード | 対応するキー | エディタ中の機能 | プログラム中の機能 |
|---|---|---|---|
| 1 | CTRL+A | HELP キーと同じ | — |
| 2 | CTRL+B | カーソルを１つ前のワードへ移動 | — |
| 3 | CTRL+C | STOP キーと同じ | — |
| 4 | CTRL+D | １ワード削除 | — |
| 5 | CTRL+E | カーソル位置から行末までを削除 | — |
| 6 | CTRL+F | カーソルを１つ先のワードへ移動 | — |
| 7 | CTRL+G | スピーカを鳴らす | スピーカを鳴らす |
| 8 | CTRL+H | BS キーと同じ | バックスペース |
| 9 | CTRL+I | TAB キーと同じ | 水平タブ |
| 10 | CTRL+J | 行の連結(インサートモードでは行の分割) | ラインフィード |
| 11 | CTRL+K | カーソルをホームポジションへ移動 | ホームポジション |
| 12 | CTRL+L | テキスト画面のクリア | テキスト画面のクリア |
| 13 | CTRL+M | リターンキーと同じ | キャリッジリターン |
| 15 | CTRL+O | 画面への表示の無効／有効の切り換え | — |
| 18 | CTRL+R | インサートモードに入る／出る | — |
| 19 | CTRL+S | 表示(スクロール)の一時停止 | — |
| 21 | CTRL+U | １行削除 | — |
| 24 | CTRL+X | カーソルを１行の最後へ移動 | — |
| 28 | → | カーソルを右へ移動 | カーソルを右へ移動 |
| 29 | ← | カーソルを左へ移動 | カーソルを左へ移動 |
| 30 | ↑ | カーソルを上へ移動 | カーソルを上へ移動 |
| 31 | ↓ | カーソルを下へ移動 | カーソルを下へ移動 |

**備考**　プログラム中の機能は，対応するコードが画面に出力される際(プログラムの実行中)に働きます．

# C. キャラクタコード表

上位4ビット→　下位4ビット↓

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | $D_E$ | | 0 | @ | P | | p | | | | ー | タ | ミ | | |
| 1 | $S_H$ | $D_1$ | ! | 1 | A | Q | a | q | | | 。 | ア | チ | ム | | 円 |
| 2 | $S_X$ | $D_2$ | " | 2 | B | R | b | r | | | 「 | イ | ツ | メ | | 年 |
| 3 | $E_X$ | $D_3$ | # | 3 | C | S | c | s | | | 」 | ウ | テ | モ | | 月 |
| 4 | $E_T$ | $D_4$ | $ | 4 | D | T | d | t | | | 、 | エ | ト | ヤ | | 日 |
| 5 | $E_Q$ | $N_K$ | % | 5 | E | U | e | u | | | ・ | オ | ナ | ユ | | 時 |
| 6 | $A_K$ | $S_N$ | & | 6 | F | V | f | v | | | ヲ | カ | ニ | ヨ | | 分 |
| 7 | $B_L$ | $E_B$ | ' | 7 | G | W | g | w | | | ァ | キ | ヌ | ラ | | 秒 |
| 8 | $B_S$ | $C_N$ | ( | 8 | H | X | h | x | | | ィ | ク | ネ | リ | ♠ | |
| 9 | $H_T$ | $E_M$ | ) | 9 | I | Y | i | y | | | ゥ | ケ | ノ | ル | ♥ | |
| A | $L_F$ | $S_B$ | * | : | J | Z | j | z | | | ェ | コ | ハ | レ | ♦ | |
| B | $H_M$ | $E_C$ | + | ; | K | [ | k | { | | | ォ | サ | ヒ | ロ | ♣ | |
| C | $C_L$ | → | , | < | L | ¥ | l | \| | | | ャ | シ | フ | ワ | ● | |
| D | $C_R$ | ← | − | = | M | ] | m | } | | | ュ | ス | ヘ | ン | ○ | |
| E | $S_O$ | ↑ | . | > | N | ^ | n | ~ | | | ョ | セ | ホ | ゛ | | |
| F | $S_I$ | ↓ | / | ? | O | _ | o | | | | ッ | ソ | マ | ゜ | | |

# D. 誘導関数

関数として用意していない三角関数のうち，いくつかは，用意された関数を使って作り出すことができます．以下の数式を参考にしてください(誤差の範囲に注意が必要です)．

| 目的とする関数 | 組み込み関数からの誘導式 |
|---|---|
| セカント | SEC(X)＝1/COS(X) |
| コセカント | CSC(X)＝1/SIN(X) |
| コタンジェント | COT(X)＝1/TAN(X) |
| アークサイン | ARCSIN(X)＝ATN(X/SQR(−X＊X＋1)) |
| アークコサイン | ARCCOS(X)＝−ATN(X/SQR(−X＊X＋1))＋1.5708 |
| アークセカント | ARCSEC(X)＝ATN(SQR(X＊X−1))＋(SGN(X)−1)＊1.5708 |
| アークコセカント | ARCCSC(X)＝ATN(1/SQR(X＊X−1))＋(SGN(X)−1)＊1.5708 |
| アークコタンジェント | ARCCOT(X)＝−ATN(X)＋1.5708 |
| ハイパーボリック・サイン | SINH(X)＝(EXP(X)−EXP(−X))/2 |
| ハイパーボリック・コサイン | COSH(X)＝(EXP(X)＋EXP(−X))/2 |
| ハイパーボリック・タンジェント | TANH(X)＝−EXP(−X)/(EXP(X)＋EXP(−X))＊2＋1 |
| ハイパーボリック・セカント | SECH(X)＝2/(EXP(X)＋EXP(−X)) |
| ハイパーボリック・コセカント | CSCH(X)＝2/(EXP(X)−EXP(−X)) |
| ハイパーボリック・コタンジェント | COTH(X)＝EXP(−X)/(EXP(X)−EXP(−X))＊2＋1 |
| ハイパーボリック・アークサイン | ARCSINH(X)＝LOG(X＋SQR(X＊X＋1)) |
| ハイパーボリック・アークコサイン | ARCCOSH(X)＝LOG(X＋SQR(X＊X−1)) |
| ハイパーボリック・アークタンジェント | ARCTANH(X)＝LOG((1＋X)/(1−X))/2 |
| ハイパーボリック・アークセカント | ARCSECH(X)＝LOG((SQR(−X＊X＋1)＋1/X) |
| ハイパーボリック・アークコセカント | ARCCSCH(X)＝LOG((SGN(X)＊SQR(X＊X＋1)＋1)/X) |
| ハイパーボリック・アークコタンジェント | ARCCOTH(X)＝LOG((X＋1)/(X−1))/2 |

# E. 予約語一覧

これらの予約語を，変数名やラベル名として使用することはできません．ただし，予約語を含む文字列（例：ABSN）は，使用してもかまいません．

| | | | |
|---|---|---|---|
| ABS | CONSOLE | EOF | INPUT$ |
| AKCNV$ | CONT | EQV | INSTR |
| ALLOC | COPY | ERASE | INT |
| AND | COS | ERL | IRESET |
| ASC | CSNG | ERR | ISET |
| ATN | CSRLIN | ERROR | JIS$ |
| ATTR$ | CVD | EXP | KACNV$ |
| AUTO | CVI | FIELD | KANJI |
| BEEP | CVS | FILES | KEXT$ |
| BLOAD | DATA | FIX | KEY |
| BSAVE | DATE$ | FN | KILL |
| CALL | DEF | FOR | KINPUT |
| CDBL | DEFDBL | FRE | KINSTR |
| CHAIN | DEFINT | GET | KLEN |
| CHDIR | DEFSNG | GOSUB | KMID$ |
| CHILD | DEFSTR | GOTO | KNJ$ |
| CHR$ | DELETE | GO TO | KPLOAD |
| CINT | DELIM | HELP | KTYPE |
| CIRCLE | DIM | HEX$ | LEFT$ |
| CLEAR | DRAW | IEEE | LEN |
| CLOSE | DSKF | IF | LET |
| CLS | EDIT | IFC | LFILES |
| CMD | ELSE | IMP | LINE |
| COLOR | END | INKEY$ | LIST |
| COM | ENVIRON | INP | LLIST |
| COMMON | ENVIRON$ | INPUT | LOAD |

LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MAIL
MAP
MERGE
MID$
MKD$
MKDIR
MKI$
MKS$
MOD
MOUSE
NAME
NEXT
NEW
NOT
OCT$
OFF
ON
OPEN
OPTION
OR
OUT
PAINT
PEEK
PCAL$
PCHK
PCNV
PCNV$
PLAY
POINT
POKE
POLL
POS
PPOLL
PPR
PRESET
PRINT
PSET
PUT
RANDOMIZE
RBYTE
READ
REM
REN
RENUM
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
ROLL
RSET
RUN
SAVE
SCREEN
SCROLL
SEARCH
SEG
SEGPTR
SET
SGN
SIN
SPACE$
SPC
SQR
SRQ
STATUS
STEP
STOP
STR$
STRING$
SWAP
SYSTEM
TAB
TAN
THEN
TIME$
TIMEOUT
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VIEW
VOICE
WAIT
WBYTE
WEND
WHILE
WIDTH
WINDOW
WRITE
XOR

# F. 1バイト/2バイトJISコード変換表

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | /■<br>2223 | DE/■<br>2223 | <br>2121 | 0<br>2330 | @<br>2177 | P<br>2350 | /■<br>2223 | p<br>2370 | ▁/■<br>2223 | ┴/■<br>2223 | /■<br>2223 | ー<br>213C | タ<br>253F | ミ<br>255F | ═/■<br>2223 | ×/■<br>2223 |
| **1** | SH/■<br>2223 | D1/■<br>2223 | !<br>212A | 1<br>2331 | A<br>2341 | Q<br>2351 | a<br>2361 | q<br>2371 | ▂/■<br>2223 | ┬/■<br>2223 | 。<br>2123 | ア<br>2522 | チ<br>2541 | ム<br>2560 | ╞/■<br>2223 | 円<br>315F |
| **2** | SX/■<br>2223 | D2/■<br>2223 | "<br>2149 | 2<br>2332 | B<br>2342 | R<br>2352 | b<br>2362 | r<br>2372 | ▃/■<br>2223 | ┤/■<br>2223 | 「<br>2156 | イ<br>2524 | ツ<br>2544 | メ<br>2561 | ╪/■<br>2223 | 年<br>472F |
| **3** | EX/■<br>2223 | D3/■<br>2223 | #<br>2174 | 3<br>2333 | C<br>2343 | S<br>2353 | c<br>2363 | s<br>2373 | ▄/■<br>2223 | ├/■<br>2223 | 」<br>2157 | ウ<br>2526 | テ<br>2546 | モ<br>2562 | ╡/■<br>2223 | 月<br>376E |
| **4** | ET/■<br>2223 | D4/■<br>2223 | $<br>2170 | 4<br>2334 | D<br>2344 | T<br>2354 | d<br>2364 | t<br>2374 | ▅/■<br>2223 | ￣/■<br>2223 | 、<br>2122 | エ<br>2528 | ト<br>2548 | ヤ<br>2564 | ◢/■<br>2223 | 日<br>467C |
| **5** | EQ/■<br>2223 | NK/■<br>2223 | %<br>2173 | 5<br>2335 | E<br>2345 | U<br>2355 | e<br>2365 | u<br>2375 | ▆/■<br>2223 | ─/■<br>2223 | ・<br>2126 | オ<br>252A | ナ<br>254A | ユ<br>2566 | ◣/■<br>2223 | 時<br>3B7E |
| **6** | AK/■<br>2223 | SN/■<br>2223 | &<br>2175 | 6<br>2336 | F<br>2346 | V<br>2356 | f<br>2366 | v<br>2376 | ▇/■<br>2223 | │/■<br>2223 | ヲ<br>2572 | カ<br>252B | ニ<br>254B | ヨ<br>2568 | ◥/■<br>2223 | 分<br>4A2C |
| **7** | BL/■<br>2223 | EB/■<br>2223 | '<br>2147 | 7<br>2337 | G<br>2347 | W<br>2357 | g<br>2367 | w<br>2377 | █/■<br>2223 | ▕/■<br>2223 | ァ<br>2521 | キ<br>252D | ヌ<br>254C | ラ<br>2569 | ◤/■<br>2223 | 秒<br>4943 |
| **8** | BS/■<br>2223 | CN/■<br>2223 | (<br>214A | 8<br>2338 | H<br>2348 | X<br>2358 | h<br>2368 | x<br>2378 | ▏/■<br>2223 | ┌/■<br>2223 | ィ<br>2523 | ク<br>252F | ネ<br>254D | リ<br>256A | ♠/■<br>2223 | /■<br>2223 |
| **9** | HT/■<br>2223 | EM/■<br>2223 | )<br>214B | 9<br>2339 | I<br>2349 | Y<br>2359 | i<br>2369 | y<br>2379 | ▎/■<br>2223 | ┐/■<br>2223 | ゥ<br>2525 | ケ<br>2531 | ノ<br>254E | ル<br>256B | ♥/■<br>2223 | /■<br>2223 |
| **A** | LF/<br>変換しない | SB/■<br>2223 | *<br>2176 | :<br>2127 | J<br>234A | Z<br>235A | j<br>236A | z<br>237A | ▍/■<br>2223 | └/■<br>2223 | ェ<br>2527 | コ<br>2533 | ハ<br>254F | レ<br>256C | ♦/■<br>2223 | /■<br>2223 |
| **B** | HM/■<br>2223 | EC/<br>変換しない | +<br>215C | ;<br>2128 | K<br>234B | [<br>214E | k<br>236B | {<br>2150 | ▌/■<br>2223 | ┘/■<br>2223 | ォ<br>2529 | サ<br>2535 | ヒ<br>2552 | ロ<br>256D | ♣/■<br>2223 | /■<br>2223 |
| **C** | CL/■<br>2223 | →<br>222A | ,<br>2124 | <<br>2163 | L<br>234C | ¥<br>216F | l<br>236C | ¦<br>2143 | ▋/■<br>2223 | ╭/■<br>2223 | ャ<br>2563 | シ<br>2537 | フ<br>2555 | ワ<br>256F | ●<br>217C | /■<br>2223 |
| **D** | CR/<br>変換しない | ←<br>222B | −<br>215D | =<br>2161 | M<br>234D | ]<br>214F | m<br>236D | }<br>2151 | ▊/■<br>2223 | ╮/■<br>2223 | ュ<br>2565 | ス<br>2539 | ヘ<br>2558 | ン<br>2573 | ○<br>217B | /■<br>2223 |
| **E** | SO/■<br>2223 | ↑<br>222C | <br>2125 | ><br>2164 | N<br>234E | ^<br>2130 | n<br>236E | ~<br>2141 | ▉/■<br>2223 | ╰/■<br>2223 | ョ<br>2567 | セ<br>253B | ホ<br>255B | ゛<br>212B | ╱/■<br>2223 | /■<br>2223 |
| **F** | SI/■<br>2223 | ↓<br>222D | /<br>213F | ?<br>2129 | O<br>234F | _<br>2132 | o<br>236F | /■<br>2223 | ┼/■<br>2223 | ╯/■<br>2223 | ッ<br>2543 | ソ<br>253D | マ<br>255E | ゜<br>212C | ╲/■<br>2223 | /■<br>2223 |

# 索引

## U



## V



## W



## X



## あ



## か



## さ



## た



## は



## ま



## や



## ら

## わ



## 記号

たしかな技術で世界をむすぶ

NEC